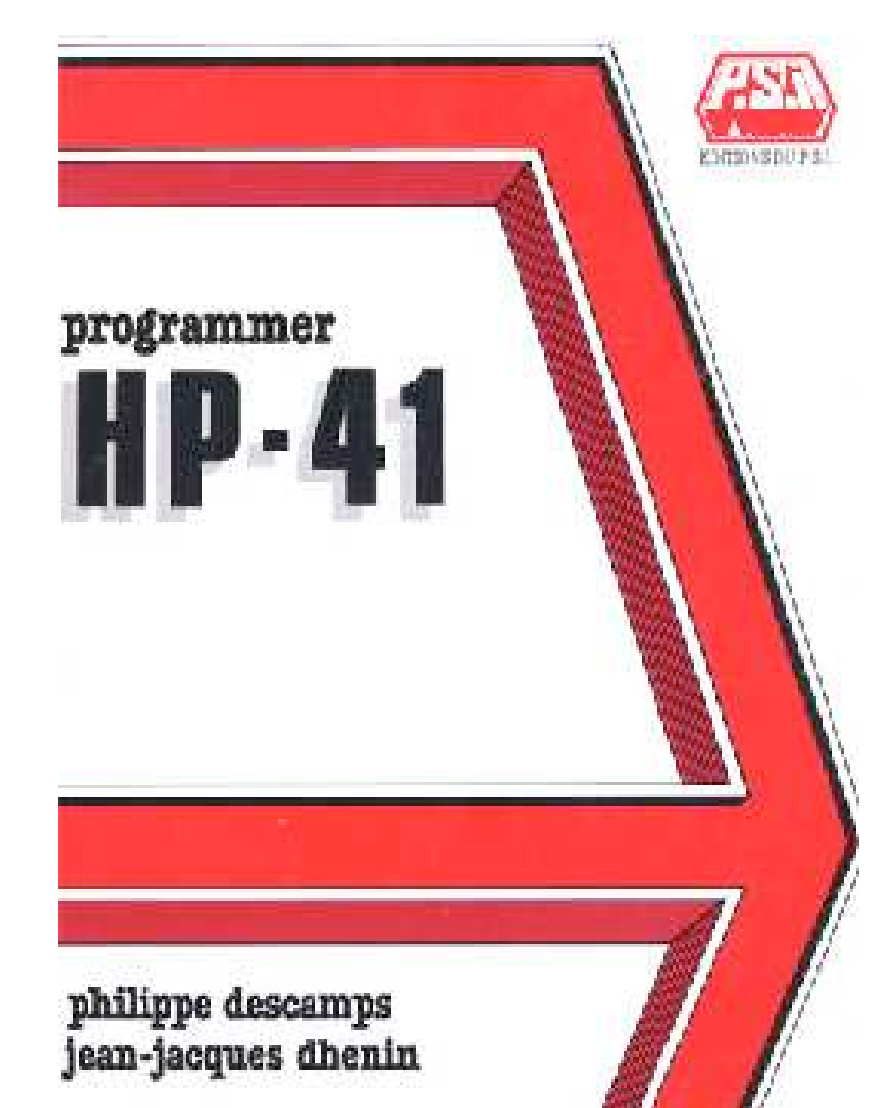

programmer
HP-41
philippe descamps
jean-jacques dhenin

Je ne me suis pas contenté
de scanner le livre.
Je l'ai repris avec Latex
pour améliorer la lisibilité

Vous pouvez contribuer à
ce travail en faisant
un don à

Dhénin Jean-Jacques
48 rue de la Justice
78300 Poissy

dhenin@dhenin.fr

JJ Dhénin

Programmer HP-41

par

Philippe Descamps

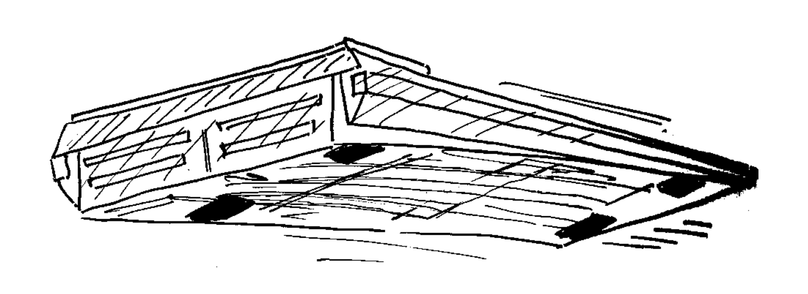
et

Jean-Jacques Dhénin



Édition du P.S.I.

1982




WD

S O M M A I R E

		Pages
Introduction		7
Chapitres :		
I	- Le langage symbolique	9
II	- Test et drapeaux	17
III	- Face à la pile	37
IV	- Hasard nécessaire	63
V	- Sur tous les tableaux	67
VI	- L'alpha et l'oméga	95
Annexes :		
1	- Solution des exercices	115
2	- Obtention des fonctions	119
3	- Description des fonctions	123
4	- Drapeaux	134
5	- Procédures avec code barres	136

INTRODUCTION

POURQUOI LIRE CE LIVRE ?

- Parce que vous possédez un HP-41 et que vous désirez tirer le meilleur parti de votre investissement ;
- parce que, ayant lu le manuel vous vous interrogez encore pour savoir si vous pourrez programmer la belle application dont vous rêvez et que vous utiliserez quotidiennement ;
- parce que vous avez peu de temps et que vous désirez devenir rapidement efficace sans trop vous fatiguer ;
- parce que vous êtes affectés par le discours abusif des "parlant Basic" et que vous souhaitez être rassurés sur la puissance du HP-41 ;
- parce que vous êtes curieux.

COMMENT UTILISER CE LIVRE ?

Si vous êtes **studieux**, nous vous proposons le processus suivant :

1. Poser N=I
2. Lisez la page N
3. *Si* vous avez tout compris *alors* N=N+I
4. *Si* le livre est terminé *alors* arrêter *sinon* continuer à l'instruction 2.

Si vous êtes **primesautier** et que l'algorithme précèdent vous effraye, ouvrez vite le livre au hasard et feuilletez-le jusqu'à ce que vous ayez fait une découverte, ce qui ne saurait tarder.

Si vous êtes **pressé**, les index et les annexes vous permettront de retrouver rapidement l'information dont vous avez un urgent besoin, et de sortir de l'incertitude qui nuit à la profondeur de votre sommeil.

Si vous êtes **paresseux**, recopiez bestialement les procédures qui fleurissent dans ces pages. Avec un peu de chance vous en trouverez une qui vous conviendra.

Si vous êtes **indépendant**, oubliez les conseils qui précèdent et décidez vous-même de la manière dont vous consulterez cet ouvrage.

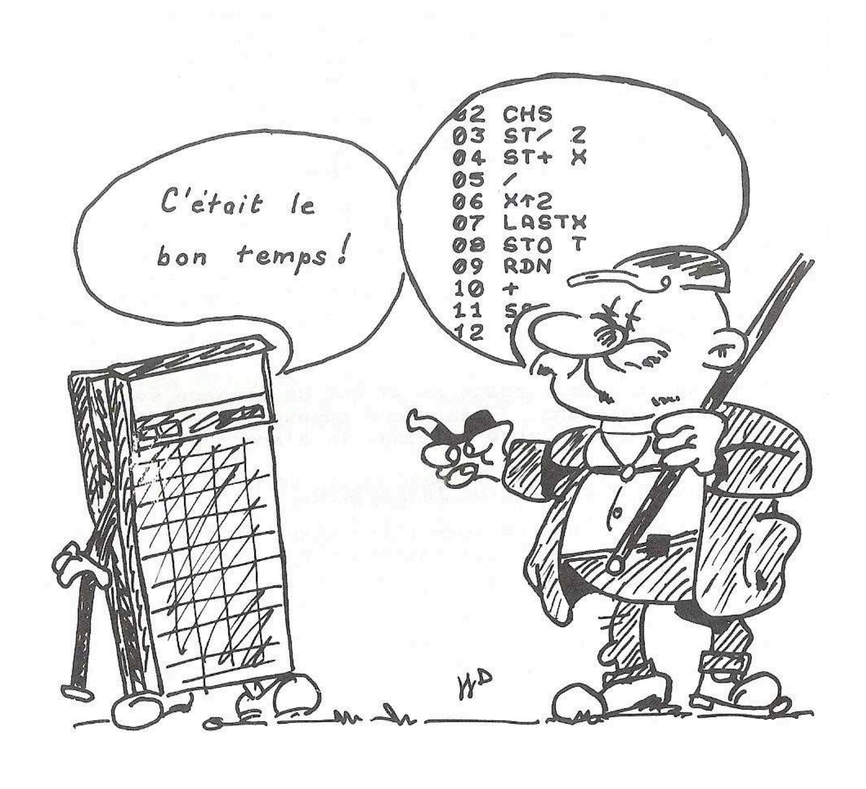
Quelle que soit la méthode adoptée, travaillez toujours avec `HP-41` à vos côtés.

Les programmes présentés n'excèdent généralement pas quarante instructions. Ils sont donc rapidement introduits au clavier. Suivez-en pas à pas l'exécution (fonction **SST**) et utilisez au maximum la fonction **AVIEW** pour vérifier constamment le contenu des différents registres.

Notre but (ambitieux) est d'être à la fois instructif et distrayant. Programmer est un plaisir que nous souhaitons vous faire partager.

CHAPITRE I

LE LANGAGE SYMBOLIQUE



Lorsqu'en septembre 1979 le `HP-41` apparut en France, il représentait un étonnant saut qualitatif dans le domaine des ordinateurs de poche. Le premier de sa catégorie il autorisait l'introduction, le traitement et l'affichage de chaînes de caractères. La voie était ouverte pour un dialogue complet entre le calculateur et son utilisateur, dialogue que favorisait encore les possibilités sonores du système.

Aussi importantes que soient ces caractéristiques elles ne représentent que la partie immergée de l'iceberg. L'existence d'un clavier et d'un affichage alphabétique conduit à d'autres caractéristiques, certaines immédiates, et d'autres plus profondes et plus riches liées à l'architecture du calculateur.

La première caractéristique est l'**affichage "en clair"**, et non plus en code, des **fonctions programmées**. L'appellation de "langage machine spécialisé", utilisé jusqu'ici pour les calculatrices programmables, est maintenant inadéquat. Tout juste peut-on parler, pour le `HP-41`, de langage d'assemblage spécialisé. En fait, en développant ce calculateur Hewlett-packard a élaboré un **langage évolué** original qui lui confère un ensemble de propriétés qui restent, à l'heure où nous écrivons ces lignes, uniques.

De la compréhension de ces possibilités dépendra la façon dont vous utiliserez votre `HP-41` : comme une calculette hypertrophiée ou comme un ordinateur puissant dont la souplesse vous évitera de nombreuses heures de navigation à l'estime. Vous pourrez (devrez !) structurer vos programmes.

Affirmer que le langage `HP-41` est un langage évolué surprendra certains. Il est donc nécessaire d'étayer cette thèse d'un nombre raisonnable d'arguments.

CE LANGAGE EST UN LANGAGE INTERPRÉTÉ

Les instructions que vous introduisez dans la mémoire de la machine ne sont pas immédiatement compréhensibles par le microprocesseur. Avant d'être exécutées elles doivent être traduites et décomposées en une série (parfois très complexe) de micro-instructions qui peuvent, elles, être appréhendées par la "puce" du `HP-41` cette opération de décryptage se nomme l'interprétation.



À titre d'exemple l'exécution de **RCL 30** (rappel en **X** du contenu du **30**ème registre de donnée) va déclencher une impressionnante cascade d'opérations :

- Calcul de l'adresse réelle du registre **30**, dont la localisation dépend de la taille de la mémoire.
- Vérification de l'existence de ce registre, avec le cas échéant l'affichage du message "**NONEXISTENT**" et arrêt de l'exécution (si le drapeau **25** n'est pas armé).
- Si la montée de la pile opérationnelle n'est pas autorisée par l'instruction qui précède (**CLX**, Σ+, Σ−) il y a alors chargement du contenu du registre adressé dans **X** , sinon s'effectue un décalage préalable de la pile vers le haut avant l'exécution de cette opération.

Que vous ayez suivi ou non le détail de la manœuvre (que nous avons outrageusement simplifiée) vous êtes à même de percevoir la différence entre une instruction en langage évolué (**RCL 30**) et le langage machine (tous les calculs et transferts de valeurs que le microprocesseur vient d'exécuter pour vous).

LE HP-41 DISPOSE D'UN LANGAGE SYMBOLIQUE

Un ordinateur, quel qu'il soit, occupe les trois quarts de son temps à rechercher des informations dans un coin de sa mémoire pour les transférer ailleurs. Il se peut qu'au passage ces informations soient modifiées mais ce

n'est pas absolument nécessaire !

Pour effectuer ces transferts le microprocesseur doit connaître les positions de départ et d'arrivée des informations, ce que l'on nomme les adresses absolues.

Il existe deux types d'informations :

– Les *données* : valeurs numériques ou groupe de caractères ;

– Les *instructions*, dont 1'enchaînement constitue un programme.

Au niveau du langage machine ces adresses sont toutes numériques. Mais l'utilisateur moyen n'a rien d'un microprocesseur, il lui est plus facile de retenir des mots que des numéros : les programmeurs préfèrent les symboles aux nombres.

Le microprocesseur doit donc effectuer la tâche fastidieuse consistant à mettre en correspondance le symbole et l'adresse absolue qui lui permettra d'atteindre 1'information désignée. Il peut le faire par l'intermédiaire de catalogues comparables, dans le principe, à l'agenda qui vous permet quotidiennement de retrouver un n° de téléphone à partir du nom d'un individu.

On peut donc mesurer le degré d'évolution d'un langage par son degré de symbolisme. Afin de mieux situer le langage HP-41 nous reproduisons le tableau donné par DANIEL-JEAN DAVID, dans "PROGRAMMER EN PASCAL" (Éditions du P.S.I.), en le complétant quelque peu :

OBJETS	**LANGAGES**				
	LANGAGE MACHINE	**BASIC**	**FORTAN**	**HP-41**	**PASCAL**
VARIABLES	NUM	SYM	SYM	NUM	SYM
SOUS-PROGRAMMES	NUM	NUM	SYM	SYM	SYM
ÉTIQUETTES	NUM	NUM	NUM	SYM	NUM
CONSTANTES	NUM	NUM	NUM	NUM	SYM

On constate que le langage HP-41 y occupe une position originale non dépourvue d'intérêt.

Dire que les sous-programmes et les étiquettes sont symboliques signifie que l'on peut écrire la séquence suivante d'instructions :

```
01*LBL  "JEU"
02 XEQ  "INIT"
03*LBL  "TOUR"
04 XEQ  "JOUEUR1"
05 XEQ  "FIN?"
06 XEQ  "JOUEUR2"
07 XEQ  "FIN?"
08 GTO  "TOUR"
09 END
```

Ce qui, sans commentaire additionnel est incomparablement plus explicite que l'équivalent Basic :

```
10 GOSUB 500
20 GOSUB 1000
30 GOSUB 3000
40 GOSUB 2000
50 GOSUB 3000
60 GOTO 20
79 END
```

LE LANGAGE DU HP-41 EST MODULAIRE

Dans le monde de la micro-informatique HP-41 est la seule machine qui autorise 1'implantation en mémoire d'un nombre indéterminé de programmes. Chacun peut être créé, modifié, effacé et (si on dispose de périphérique de masse) lu ou écrit indépendamment.

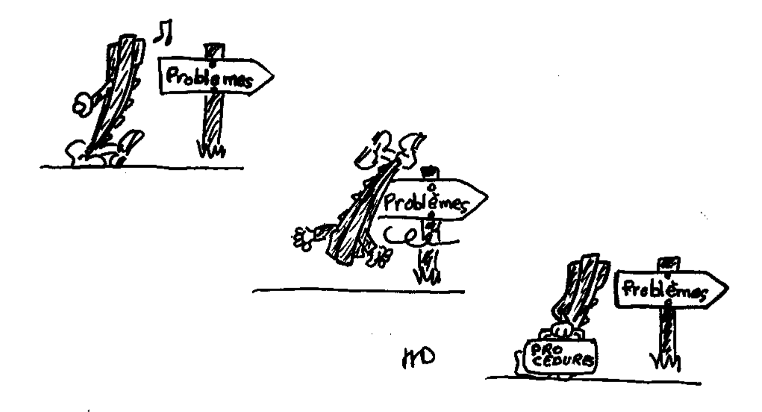
À cette indépendance physique s'oppose une dépendance logique. Tout programme peut commander l'exécution d'un groupe d'instructions appartenant à un autre programme il suffit que cette séquence débute par une étiquette alphabétique : **LBL "X..."** et s'achève par **RTN** ou **END**.

Il est alors possible de décomposer un programme complexe en une série de sous-unités (elles-mêmes décomposables en blocs plus simples) selon les sains principes de la programmation descendante. Les problèmes posés par la manipulation élémentaire des données sont ainsi renvoyés au niveau d'appels les

plus profonds et ne viennent plus noyer 1'architecture logique du programme clairement détaillée aux niveaux supérieurs.

Cette façon de faire offre de multiples avantages : Il devient très difficile de commettre des erreurs de conception dans l'agencement des opérations nécessaires à la résolution d'un problème. Si malgré tout, cela advenait, cette erreur serait rapidement localisée parmi un nombre restreint d'instructions.

Il est également possible de tester individuellement chacun des blocs pour vérifier que les sorties obtenues correspondent bien aux valeurs espérées pour un jeu de données d'essais proposées en entrée.



En conclusion, `HP-41` dispose d'un langage interprété puissant. Chaque problème peut être décomposé en sous-unités indépendantes facilement programmables, chacune est désignée par un nom symbolique qui décrit sa fonction et simplifie son repérage.

PROGRAMMATION SUR LE HP-41

Il existe deux niveaux de programmation sur `HP-41` :

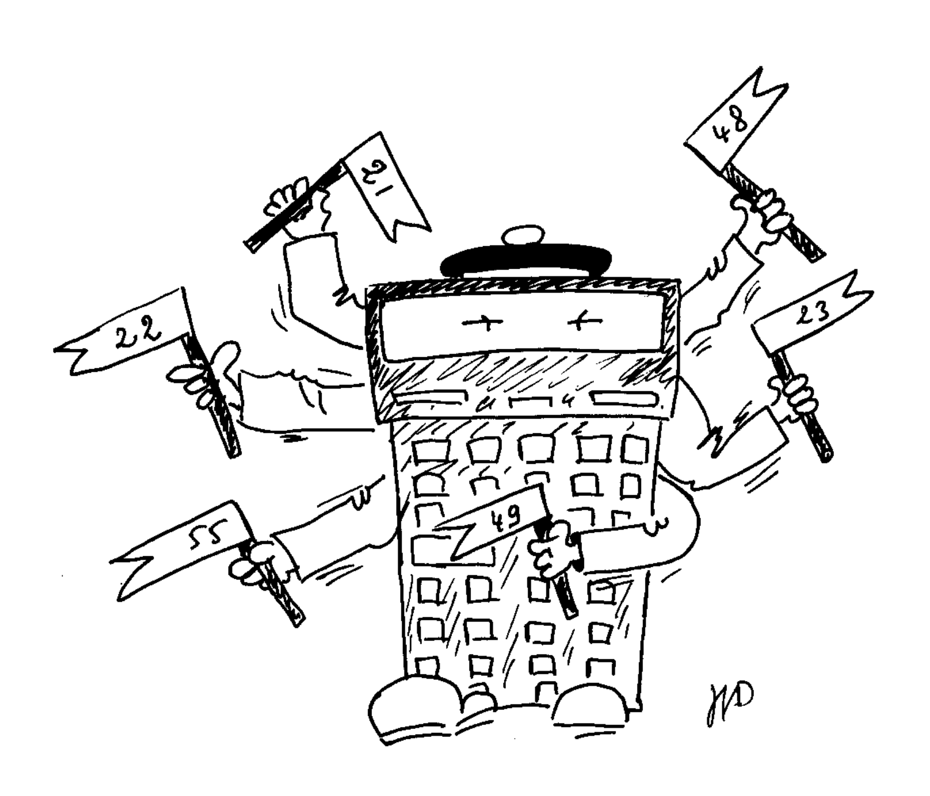
- Les **programmes proprement dits**, constitués en fait d'une succession d'appels de procédures éventuellement entrecoupés de tests. Toujours orienté vers la ré-

solution d'un problème spécifique, le programme résume la part que nous nommerons "stratégique" de l'art du programmeur. Compréhensible à la simple lecture, un programme doit avoir une documentation interne par le biais des étiquettes et des commentaires (nous verrons par la suite comment introduire des remarques dans le corps du programme).

- Les **procédures** (ou sous-programmes de traitement). Elles représentent le versant "tactique" de la programmation. Une procédure est courte, rapide, elle économise au maximum l'espace programmable et altère le moins possible 1'espace des variables. Elle correspond à l'exécution d'une tâche unique.

Elle est souvent suffisamment générale pour être utilisée de multiples fois par un même programme, voire par des programmes distincts. Elle est donc susceptible d'être maintenue en mémoire de façon permanente. Ce fait impose une standardisation des méthodes de programmation, source d'une importante économie d'effort. Si une procédure répond à tous ces critères elle peut être considérée comme une nouvelle fonction du langage `HP-41` démonstration de 1'ultime qualité de ce langage : sa plasticité, son évolutivité.

CHAPITRE II
TESTS ET DRAPEAUX



Au tout début de la rédaction de ce livre il était prévu que ce chapitre, en raison de sa relative complexité, vienne clore 1'ouvrage. Il nous est rapidement apparu qu'il était indispensable de maîtriser les concepts qu'il décrit pour pouvoir programmer HP-41 . Nous vous présentons donc sans plus tarder : les outils de la décision.

Pour les anglo-saxons le mot "computer" désigne littéralement un calculateur. Parce qu'il ne fait pas référence à la notion de calcul, 1'équivalent français, ordinateur, nous semble plus significatif.

Ordonner implique la nécessité d'effectuer des choix. À chaque instant le déroulement des opérations est déterminé par les instructions programmées mais également par les événements antérieurs. L'ordinateur est amené à prendre des décisions : il sait faire des tests.

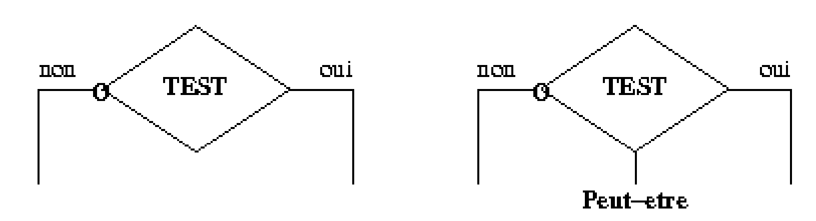
HP-41 dispose de dix tests arithmétiques et quatre tests sur les indicateurs binaires (drapeaux).

X=0?	X=Y?
X≠0?	X≠Y?
X<0?	X<Y?
X<=0?	X<=Y?
X>0?	X>Y?
FC?	FC?C
FS?	FS?C

Remarque : Les fonctions **ISG** et **DSE** ne seront pas évoquées dans ce chapitre. Elles se comportent cependant comme des tests.

Ces tests sont binaires car la réponse aux questions posées est obligatoirement **OUI** ou **NON** .

Structure générale



Si la réponse au test est oui alors l'instruction qui suit est exécutée. La phrase : « *Si* tu as un zéro de conduite, *alors* tu auras deux desserts ! » s'écrit en langage HP-41 courant :

```
01 "NOTE?"          Introduction de la note
02 PROMPT
03 "DESSERT"        Préparation de l'affichage
04 X=0?             TEST
05 "├ *2 .."        Si OUI, modification de l'affichage
06 AVIEW            Affichage
07 .END.
```

La clef d'une bonne utilisation des tests réside dans le choix de l'instruction qui suit. Ainsi dans 1'exemple qui précède une ligne de texte vient en modifier une autre transformant le sens du message. Contrairement à d'autres langages celui du HP-41 autorise à faire suivre un test par une fonction quelconque.

L'exemple qui suit illustre comment placer simplement la plus petite de deux valeurs initialement en X et Y.

```
01 X>Y?
02 X<>Y
```

Dans un troisième exemple 1'exécution du programme est interrompue

pour demander l'introduction d'une valeur si le registre X est nul :

```
01 "VALEUR?"
02 RCL 00
03 X=0?
04 PROMPT
05 STO 00
```

Remarque : Cette séquence est très utile pour initialiser un registre. L'arrêt du programme ne se produira que s'il est nécessaire. La tactique utilisée dans ce cas est de préparer une action qui, en fonction du test, sera exécutée ou non.

Bien souvent, hélas, une seule instruction est insuffisante. Il faut alors introduire une rupture de séquence en utilisant **XEQ** ou **GTO**.

L'appel d'un bloc d'instructions par* XEQ *(doit être préféré au simple branchement* GTO*. Après l'exécution de la séquence le programme continue en séquence, facilitant sa lecture.

L'instruction **GTO** doit être réservée au cas où le programme diverge en deux "pavés" terminaux totalement distincts.

LES DRAPEAUX ET LA DESCRIPTION DE L'ENVIRONNEMENT

HP-41 est un automate complexe. C'est un système dont le comportement est totalement déterminé par construction. Ses transitions d'un état à un autre sont reproductibles ; soumis à une même excitation il réagit toujours de la même façon.

C'est essentiellement dans l'utilisation des drapeaux que cet aspect présente un intérêt. HP-41 dispose du nombre confortable de **56** drapeaux qui lui permettent de mémoriser et d'analyser ce que nous appellerons l'environnement logique.

DES DRAPEAUX POUR COMMÉMORER L'ÉVÉNEMENT

Un drapeau est essentiellement une mémoire mais, contrairement aux registres de données, ce ne sont pas des valeurs numériques qui sont stockées mais des événements.

Chaque indicateur peut prendre deux valeurs logiques :

– il peut être **armé** (en anglais : **SET**) indiquant que l'événement a eu lieu ;
– il peut être **désarmé** (en anglais : **CLEAR**) si 1'événement ne s'est pas produit.

Même si, et ce serait regrettable, vous n'utilisez jamais les drapeaux, HP-41 les manipule lui-même pour son propre compte.

Parmi les **56** indicateurs disponibles, **26** sont gérés uniquement par la machine, **19** sont partagés entre elle et vous. Enfin les onze premiers (de 00 à 10) vous appartiennent en propre.

Pour avoir un panorama complet des utilisations possibles des drapeaux le mieux est d'observer comment les manipule un orfèvre en la matière : le calculateur lui-même.

LES 26 DRAPEAUX INTERNES

Ce domaine réservé s'étend des indicateurs 30 à 55. Si vous tentez de modifier 1'état de ces drapeaux par les instructions **SF** (Set Flag) ou **CF** (Clear Flag) HP-41 vous répondra avec une mauvaise foi évidente "**NONEXISTENT**" ; pourtant chacun de ces indicateurs a une fonction bien déterminée et leur existence ne peut faire l'objet d'aucun doute.

Examinons-les en détail :

■ 30 - Ce drapeau indique au calculateur qu'une lecture de catalogue est en cours. Les fonctions du clavier sont alors redéfinies : la touche [R/S] arrête ou reprend la liste des fonctions ; [SST] ou [BST] permettent d'avancer ou de reculer dans la liste, la touche [←] fait sortir du mode catalogue. Les autres touches son inopérantes ; le seul rôle qui leur est laissé est de pouvoir servir de "frein" pour ralentir la vitesse du défilement. Notons, au passage, que cette possibilité de redéfinir le clavier sera exploitée encore davantage par certaines extensions comme le module TIMER. S'il est impossible de modifier l'état de ces drapeaux directement, il est par contre intéressant de les tester. En ce qui concerne l'indicateur 30, comme il n'est pas possible d'interroger le

calculateur pendant un catalogue, la réponse à la question **FS? 30** sera toujours **NON**. Ceci sera pour nous d'un grand intérêt dans les programmes, comme nous le verrons tout à l'heure.

■ 31 à 35 - Ces indicateurs sont réservés au dialogue avec les périphériques connectables par l'intermédiaire de la boucle HP-IL . Comme nous ne traiterons pas des périphériques dans ce livre ces cinq drapeaux seront par prudence ignorés.

■ 36 à 41 - Ces drapeaux servent à définir le format d'affichage des nombres. Ce sont de curieux drapeaux internes car ils sont en pratique modifiés par l'utilisateur. Ces modifications ne s'effectuent pas par le biais des fonctions **SF** ou **CF**, mais par l'intermédiaire des instructions de formatage : **FIX**, **SCI** et **ENG**. Ils illustrent parfaitement le rôle de mémoire que peuvent jouer les indicateurs binaires. Les quatre drapeaux 36 à 39 conservent le nombre de décimales à afficher, pour cela la numérotation binaire est utilisée :

N°	Valeur du bit	Nb de décimales									
		0	1	2	3	4	5	6	7	8	9
39	1	0	1	0	1	0	1	0	1	0	1
38	2	0	0	1	1	0	0	1	1	0	0
37	4	0	0	0	0	1	1	1	1	0	0
36	8	0	0	0	0	0	0	0	0	1	1

Une règle générale peut être déduite de cet exemple : pour mémoriser, à l'aide des drapeaux, un état parmi n possibles il est nécessaire d'utiliser d drapeaux avec : $d = {}^{\log n}/_{\log 2}$, d étant arrondi à l'entier supérieur. Pour dix états possibles (de 0 à 9 décimales) on obtient : ${}^{\log 10}/_{\log 2} = {}^{1}/_{0,301} = 1,585$. D'où $d = 4$

■ 40 et 41 - Comme les quatre drapeaux précédents ils ne sont jamais modifiés par le calculateur. Leur rôle est de définir le mode **FIX**, **SCI** ou **ENG** de l'affichage en

fonction du code :

N°	SCI	FIX	ENG
40	0	0	1
	0	1	0

Comme il existe trois modes d'affichage, la règle énoncée ci-dessus confirme que deux drapeaux sont nécessaires.

Si les indicateurs **36** à étaient accessibles à 1'utilisateur, définir un affichage scientifique avec trois décimales pourrait se faire grâce à la séquence :

CF 36 CF 37 SF 38 SF 39 (Nombre de décimales)

CF 40 CF 41 (Mode scientifique)

On comprend que les concepteurs du HP-41 aient préféré offrir à 1'utilisateur les fonctions **FIX**, **ENG** et **SCI**.

■ 42 et 43 - Ce sont encore des drapeaux modifiables par le biais d'instructions spéciales pour définir 1'environnement de travail du HP-41 . Ils précisent quelle est 1'unité d'angle utilisée. Trois unités étant disponibles DEGRE, RADIAN ou GRADE deux drapeaux sont encore nécessaires pour mémoriser le mode courant suivant un code comparable à celui utilisé pour le mode d'affichage :

N°	DEG	RAD	GRAD
42	0	0	1
43	0	1	0

Le mode d'affichage ou 1'unité d'angle choisie sont clairement visibles à l'affichage, mais un programme qui s'exécute ne peut consulter cet affichage. Nous vous proposons donc une courte séquence qui insérée dans un programme

vous permettra de connaître le mode actuellement sélectionné :

```
01 0
02 FC? 42
03 SIGN
04 FC? 43
05 ST+ X
```

En fin d'exécution le registre X contient **0** si l'unité d'angle est le grade, la fonction **SIGN** n'a pas été exécutée et le doublement du registre X par la fonction **STO+X** n'a pas modifié grand chose.

Si 1'unité est le radian le contenu de X est **1**, la fonction **SIGN** étant cette fois prise en compte mais pas **STO+X**. Enfin dans le cas où il s'agit de degré, les deux fonctions qui suivent les tests donnent **2** comme résultat final.

La même séquence, légèrement modifiée, vous permettra d'obtenir sous forme de chiffre de **0** à **2** le format d'affichage courant, nous vous laissons le soin de 1'établir.

■ 44 - Ce drapeau est remis à zéro par le calculateur au moment de la mise sous tension. Il est armé par l'instruction **ON**. Son rôle, modeste, est d'éviter 1'extinction du calculateur après les fatidiques dix minutes d'inactivité. Une fois levé le seul moyen de le désarmer est d'éteindre le calculateur. En programmation il représente donc le cas très particulier d'un indicateur à usage unique !

■ 45 à 47 - Ces trois drapeaux, tout comme l'indicateur **30**, sont toujours désarmés quand 1'utilisateur les teste. Ils concernent des fonctions intimes du calculateur :

Le 45 indique qu'un nombre ou une chaîne de caractères est en cours d'introduction, la chose est concrétisée à 1'écran par le tiret qui suit la valeur déjà introduite. Le comportement de la touche d'effacement est également modifié, si ce drapeau est armé seul le dernier caractère affiché sera éliminé.

L'indicateur **46** joue un rôle identique mais lors de l'introduction d'instructions nécessitant plusieurs frappes de touches.

Le drapeau **47** indique quant à lui que la touche jaune vient d'être enfoncée, il est matérialisé à 1'affichage par 1'indication **SHIFT**.



■ 48 - Cet indicateur précise si le calculateur est ou non en mode alpha. Quand ce mode est actif l'indication **ALPHA** apparaît à droite de 1'affichage, le contenu du registre ALPHA est visualisé, et le clavier est complètement redéfini, chaque touche codant maintenant pour un caractère alphabétique ou une fonction spécifique. L'état de ce drapeau peut être modifié soit en agissant sur la bascule **ALPHA** en haut du clavier, soit par les fonctions **AON** et **AXOFF**. Il peut être testé en cours de programme, enfin le calculateur le désarme à la mise sous tension ou lors du passage en mode programme.

■ 49 - L'indicateur de décharge de la batterie. Il est associé à 1'affichage de **BAT** à gauche de 1'écran.



Si vous avez à exécuter un long programme en utilisant HP-41 de façon autonome, surtout s'il est équipé d'accumulateurs et non de piles, vous pouvez placer à certains endroits stratégiques de ce programme la séquence :

```
FS? 49
OFF
```

qui éteindra le calculateur avant qu'il ne plonge de lui-même dans un profond coma avec perte complété de la mémoire "permanente".

■ 50 - Ce drapeau indique au calculateur si un message a été placé à 1'affichage par les fonctions **VIEW** ou **AVIEW**. S'il est armé, le calculateur exhibe le registre X en mode numérique, le registre ALPHA en mode alphabétique ou promène le canard (✈) à l'écran si un programme s'exécute.

■ 51 à 54 - Encore quatre drapeaux qui resteront pour vous toujours à 1'état zéro. Le calculateur, lui s'en sert pour savoir que la touche **SST** vient d'être pressée (indicateur **51**) ou se souvenir que l'on est en mode programme (indicateur **52**), dans ce cas il affiche 1'instruction sur laquelle est arrêté le pointeur programme et considère toute pression de touche comme une instruction à insérer dans le texte du programme. HP-41 peut savoir (indicateur **53**) si un périphérique est prêt à poursuivre un fructueux dialogue (l'imprimante a-t-elle vidé son tampon, une carte a-t-elle été introduite dans le lecteur ?). Enfin 1'indicateur **54** indique qu'une pause (**PSE**) s'exécute.

■ 55 - Si vous ne disposez pas d'une imprimante cet indicateur sera, pour vous, toujours désarmé. Dans le cas contraire il sera armé dès la mise sous tension de 1'appareil si un périphérique imprimant est connecté.

La description que vous venez de lire nous a paru indispensable à une bonne compréhension des mécanismes internes de HP-41. La description de ces drapeaux faite dans le manuel reste difficilement compréhensible. Nous avons tenté d'être plus explicite.

DES DRAPEAUX PLUS ACCESSIBLES

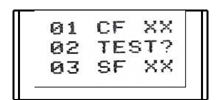
Examinons maintenant ce qu'il est possible de faire avec les indicateurs **F00** à **F29** directement accessibles à 1'utilisateur.

En s'appuyant sur la description qui précède, nous pouvons classer les applications possibles en deux catégories :

- la mémorisation d'une option à long terme. C'est le cas des drapeaux **F21** (arrêt ou non après les fonctions **VIEW** et **AVIEW**), **F24** (autorisation des dépassements en cours de calcul), **F26** (activation du signal sonore), **F27** (mode**USER**), **F28** et **F29** (point ou virgule décimale, séparateur de groupe de chiffres).
- la détection d'un événement transitoire, nécessitant généralement un traitement rapide. Entrent dans cette catégorie, les drapeaux **F22** (détection d'entrée numérique), **F23** (détection d'entrée alphabétique), **F25** F25 (détection d'erreur).

C'est dans cette seconde catégorie que se place le cas très important d'un drapeau utilisé comme variable logique pour mettre en mémoire le résultat d'un test.

Les choses se passent selon le schéma suivant :



Le drapeau choisi est préalablement désarmé, puis, en fonction du résultat du test, il est éventuellement réarmé.

Cette méthode est utile :

- quand les valeurs testées ne sont pas celles qui doivent être traitées. Entre le test initial et le traitement il faut alors modifier le contenu de la pile. Après cette modification un deuxième test, portant sur l'état du drapeau, est effectué pour tenir compte du résultat du premier ;
- quand le test est effectué dans une procédure, et le traitement résultant pris en compte par le programme appelant. Il peut s'agir en particulier

de la détection d'une erreur ;

– pour réaliser la structure SI...ALORS...SINON. On obtient dans ce cas une séquence du type :

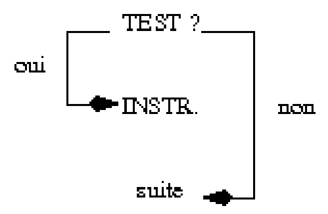
```
01 CF XX
02 TEST?
03 SF XX
04 FS? XX
05 XEQ"TRT1"
06 FC?C XX
07 XEQ"TRT2"
```

Les applications des drapeaux sont donc multiples. Il vous faut les connaître toutes si vous désirez obtenir le maximum de votre HP-41 . Et si les considérations qui précèdent ne vous ont pas trop effrayées voyez la suite...

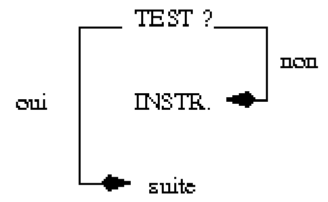


L'ensemble des exemples étudiés jusqu'à maintenant concerne l'exécution

d’une instruction si une condition unique *es*t vérifiée. Schématiquement cela correspond à :



La première question que nous aborderons est 1’exécution d’une instruction si une condition *n’est pas* vérifiée. Cette opération correspond à la négation logique. En voici le schéma :

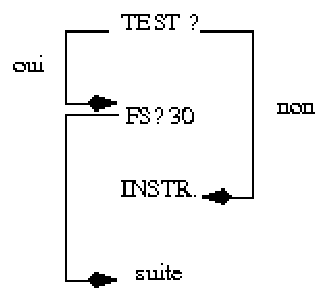


La méthode la plus immédiate pour obtenir ce résultat est de remplacer le test initial par un autre qui teste la condition inverse. Nous sommes alors ramené au cas de figure précédent pour lequel 1’instruction est exécutée si la condition est vérifiée. Le tableau qui suit donne les tests disponibles et leurs contraires :

FS?	↔	FC?
FS?C	↔	FC?C
X=0?	↔	X≠0?
X<0?	↔	Inexistant
X>0?	↔	X<=0?
X=Y?	↔	X≠Y?
X<Y?	↔	Inexistant
X>Y?	↔	X<=Y?
ISG	↔	Inexistant
DSE	↔	Inexistant

Hélas, nous constatons que la méthode n'est pas universelle. Certaines instructions de test n'ont pas de négation. Il existe cependant un procédé général pour obtenir 1'inversion d'un test, il suffit d'utiliser un "**négateur universel**".

Derrière ce nom pompeux se cache simplement un test dont la réponse est toujours négative. Nous avons adopté 1'instruction `FS? 30`. Toujours en utilisant notre notation schématique on obtient :



Si la réponse au test est négative le calculateur saute le pas suivant (`FS? 30`) et exécute l'instruction qui suit. Dans le cas contraire `FS? 30` sert littéralement de tremplin et l'instruction n'est pas exécutée. Comme exemple, la séquence :

```
01 X<0?
02 FS? 30
```

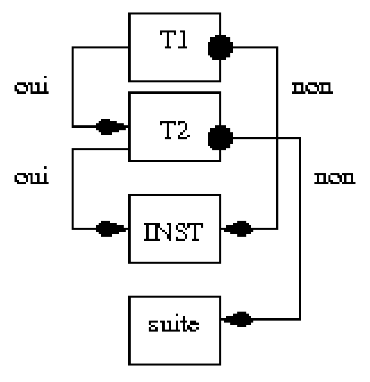
est rigoureusement équivalente à $X \geq 0?$, l'un des tests qui font défaut au calculateur.

DE PLUS EN PLUS FORT...

« *Et maintenant, Mesdames et Messieurs, nous avons le plaisir et l'avantage de vous présenter un numéro exceptionnel : l'exécution conditionnelle d'une instruction en fonction des résultats conjoints de deux tests !* » (Silence sous le chapiteau, les tambours roulent, sur la piste les artistes se concentrent).

Pour réaliser ce tour de force `HP-41` ne dispose pas (encore) d'instructions micro-programmées. Il va falloir innover.

Examinons préalablement le résultat de la mise en séquence de deux tests que nous appellerons **T1** et **T2**. Le schéma d'exécution est le suivant :



L'instruction **INST** est exécutée si :

– **T1** n'est pas vérifié. (Quel que soit le résultat de **T2**)

– **T1** et **T2** sont simultanément vérifiés.

Dans le monde de la logique Booléenne cette fonction porte le nom d'implication et admet la table de vérité suivante :

T1	T2	INST
non	non	exécutée
non	oui	exécutée
oui	non	non exécutée
oui	oui	exécutée

Transposée en langage courant cette fonction correspond à des phrases du type :

« Si le fait qu'un animal soit un batracien implique que c'est un vertébré alors les zoologistes sont des gens raisonnables ».

Dans le cas d'une grenouille la première et la seconde conditions sont vérifiées (ligne **4** de la table de la vérité) et nous ne mettrons pas en doute la raison des zoologistes.

Il en est de même pour une araignée (ligne **1**) ou un gorille (ligne **2**) qui ne sont pas des batraciens. Pour ceux-ci, "être ou non vertébré" laisse les zoologistes raisonnables.

Cependant, à la suite de notre récent voyage sur Mars, nous avons eu la

chance de capturer un KROAQ. Cette charmante bestiole est à l'évidence un batracien, mais n'a pas de vertèbre (? ?).

Lorsque nous avons présenté l'animal aux zoologistes il ne fut pas nécessaire de nous reporter à la ligne 3 de la table de vérité pour comprendre, à leurs airs hagards, que les zoologistes n'étaient plus des gens raisonnables.

Quittons notre zoo imaginaire pour revenir aux préoccupations plus terre à terre des pauvres programmeurs.

L'implication est une des fonctions logiques les plus difficiles à appréhender. I1 serait préférable de disposer, à la place, des deux fonctions fondamentales de la logique booléenne : les fonctions **ET** et **OU**.

LA FONCTION OU

Le **OU** logique est vérifié si l'une, au moins, des deux conditions qu'il relie, est vérifiée. Elle admet donc la table de vérité suivante :

T1	T2	INST
non	non	non exécutée
non	oui	exécutée
oui	non	exécutée
oui	oui	exécutée

En pratique la fonction **OU** peut se déduire de l'implication :

T1 ou **T2** peut aussi s'écrire :

NON T1 implique **T2** ou encore

NON T2 implique **T1**.

Il suffit donc pour obtenir un **OU** de prendre l'inverse de l'un des tests et de le faire suivre par le second.

Ainsi $X \geq 0?$ est équivalent à $X > 0?$ **OU** $X = 0?$, et donc à $X < 0?$ implique $X = 0?$

Nous obtenons donc deux nouveaux équivalents de notre instruction manquante :

```
01 X<0?
02 X=0?
```

et

```
01 X≠0?
02 X>0?
```

Et nous conclurons péremptoirement en affirmant que :

«*Si ce n'est pas un batracien OU si c'est un vertébré alors les zoologistes sont raisonnables*».

LA FONCTION ET

L'obtention de la fonction **ET** nous demandera un peu plus d'effort. (Le roulement de tambour s'intensifie, une légère angoisse étreint le public.)

I1 existe heureusement une relation, due à DE MORGAN qui relie les fonctions **ET** et **OU** :

T1 ET T2 = NON (NON T1 OU NON T2)

Comment faire avaler çà à HP-41 ? Avec de l'ordre et de la méthode, on peut.

Pour le lecteur attentif, en vertu de tout ce qui précède, il est évident que :

T1 ET T2 = NON (T1 implique NON T2)

Pour obtenir la négation de l'implication entre parenthèses, nous utiliserons le négateur universel.

Voyons la chose sur un exemple. Nous désirons exécuter l'instruction **BEEP** si le drapeau **01** *ET* le drapeau **02** sont simultanément armés. On pourrait

écrire :

```
01 FC? 01
02 GTO 00
03 FS? 02
04 BEEP
05*LBL 00
```

Nous préférons :

```
01 FS? 01
02 FC? 02
03 FS? 30
04 BEEP
05 .END.
```

L'exécution en est plus rapide, la séquence utilise un octet de moins et fait l'économie d'une étiquette numérique. (Coups de cymbales, musique. Sur la piste les artistes saluent pour répondre aux acclamations du public. Merci).

Mais le spectacle continue. Nous vous proposons un petit exercice pour faire la synthèse de ce chapitre.

En utilisant deux drapeaux, il faut écrire quatre procédures de telle sorte qu'aucune d'entre elles ne puisse être exécutée deux fois de suite.

La première procédure ajoute "A" au contenu du registre alpha (instruction " **A**"), la seconde ajoute "B" (" **B**"), la troisième et la quatrième ajoutent respectivement "C" et "D".

Chaque procédure se termine par **AVIEW**, **RTN**. Il sera pratique d'utiliser les étiquettes alphabétiques locales (**LBL A**, **LBL B**, **LBL C** et **LBL D**) pour désigner ces procédures.

À l'exécution vous pourrez obtenir des chaînes de caractères du type : "**ACDBDCACB**", mais pas "**ABBCAAD**" qui contient des séquences de lettres identiques.

Avant que les feux de la rampe ne se rallument pour la suite du spectacle,

un conseil. Si une instruction vous semble faire cruellement défaut, ne baissez pas les bras. Un peu de réflexion, un brin d'astuce et surtout une lecture attentive des bons auteurs, vous permettront toujours de franchir l'obstacle.

Vaincre une difficulté n'est pas perdre son temps. C'est ainsi qu'on s'élève peu à peu vers les sommets de la programmation. Un grand pas pour vous, un petit pas pour 1'humanité.

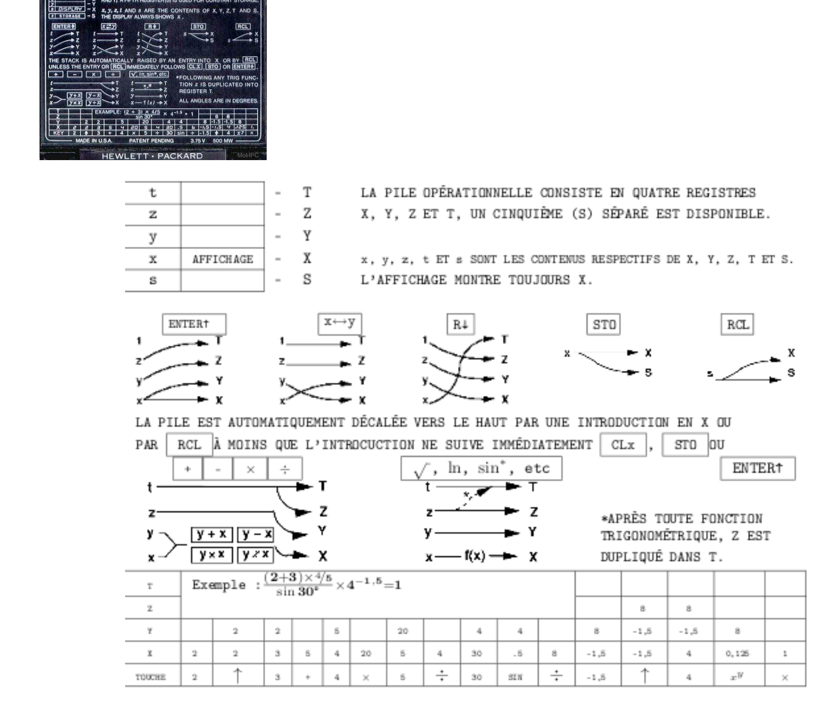
CHAPITRE III
FACE À LA PILE



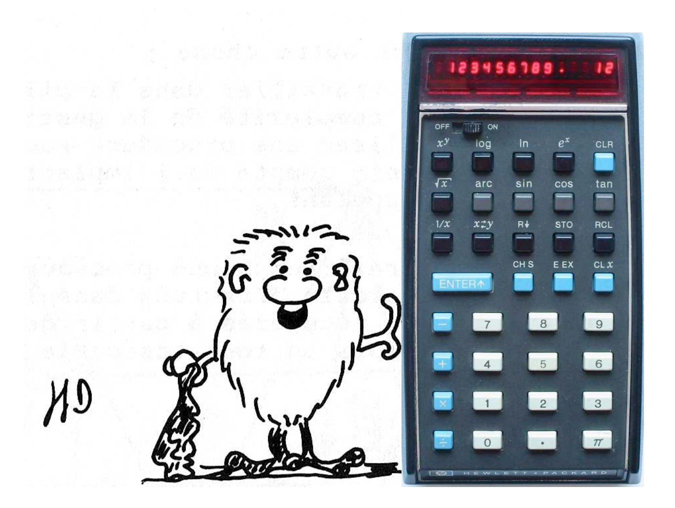
Avant d'aborder toute programmation il est essentiel de se pencher longuement sur la pile opérationnelle à quatre registres. Cette pile est le cœur du système. Toutes les valeurs doivent transiter par elle et c'est en son sein que s'effectue la majorité des opérations. Il est donc nécessaire de la maîtriser complètement et de pouvoir suivre en permanence le ballet des valeurs dans les registres.

La pile opérationnelle, associée à la notation polonaise inverse, est la constante qui, depuis leur origine, caractérise les calculateurs HEWLETT-PACKARD.

Au dos d'HP-35 se trouvait déjà un petit aide-mémoire fort susceptible de désorienter les téméraires acquéreurs de cette première calculatrice sophistiquée. Nous ne pouvons résister au plaisir de vous le présenter.



T	Exemple : $\frac{(2+3)\times 4/5}{\sin 30°}\times 4^{-1,5}=1$															
Z													8	8		
Y		2	2		5		20		4	4		8	-1,5	-1,5	8	
X	2	2	3	5	4	20	5	4	30	.5	8	-1,5	-1,5	4	0,125	1
TOUCHE	2	↑	3	+	4	×	5	÷	30	SIN	÷	-1,5	↑	4	x^y	×



Les choses ont depuis quelque peu évolué. Le stockage d'une valeur en mémoire n'inhibe plus la montée de la pile. L'opération exponentiation est maintenant $\boxed{Y^x}$ et non plus $\boxed{X^y}$, apparemment plus logique, mais en opposition complète avec l'ordre de lecture des formules. Enfin l'extension, sur `HP-41`, des opérations de rappel de stockage et d'échange, ainsi que l'arithmétique directe dans la pile, en a considérablement accru la maniabilité et la puissance.

Ce livre aurait pu s'intituler “tout faire dans la pile”. En effet, la quasi totalité des procédures que nous allons développer n'utilise que les cinq registres : `L`, `X`, `Y`, `Z`, `T` qui la constituent.

Pourquoi ce parti pris ?

Ce n'est pas uniquement par esprit sportif, pour le simple plaisir de faire “tenir dans la pile” un algorithme complexe, ce plaisir existe, sans conteste, mais l'effort accompli est rapidement rentabilisé pour trois raisons :

- une **économie de temps** : l'accès aux registres mémoire est sur ce type de calculateurs relativement lent (avant toute opération le microprocesseur doit en permanence tester l'existence des registres adressés). Cette opération ne concerne pas les registres de la pile toujours présents quelle que soit la partition choisie ;
- une **économie d'espace** : ne pas utiliser un registre pour stocker un

résultat intermédiaire, c'est le rendre disponible pour autre chose ;
- mais surtout travailler dans la pile, c'est **éviter d'accroître la complexité de la gestion de la mémoire**, et pouvoir utiliser une procédure sans avoir à la modifier pour tenir compte de l'implantation des variables du programme appelant.

Ainsi les paramètres d'une procédure sont transmis par la pile, les calculs effectués dans la pile, les résultats finalement récupérés à partir de la pile : la procédure forme alors un tout insécable, transposable, universel.

LE LOGIGRAPHE

La résolution des problèmes que nous allons aborder nécessite une utilisation optimale des mouvements de la pile. Il est donc utile de disposer d'un outil qui permette d'obtenir rapidement ces solutions optimales.

Avant de décrire cet outil et pour mieux vous en faire sentir l'intérêt, nous vous proposons un petit exercice :

À partir de la situation de droite il faut obtenir celle de gauche en n'utilisant que deux instructions

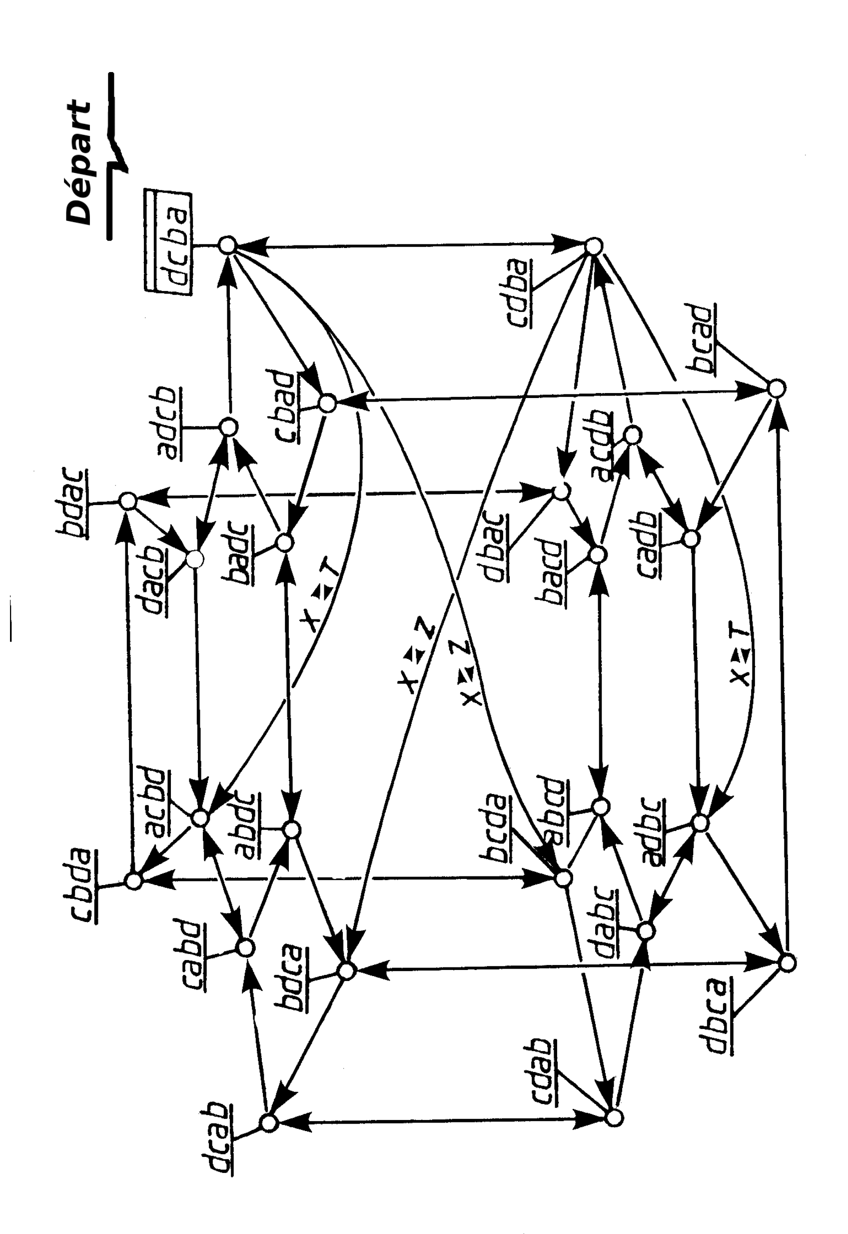
```
T = 4        T = 3
Z = 3        Z = 4
Y = 2        Y = 1
X = 1        X = 2
```

Si vous pouvez résoudre ce problème en moins de 10 secondes vous pouvez allègrement sauter la suite de ce chapitre ! Dans le cas contraire, vous tirerez profit de l'utilisation du *Logigraphe.*

Considérons nos quatre registres dans la position initiale suivante :

T = A
Z = B
Y = C
X = D

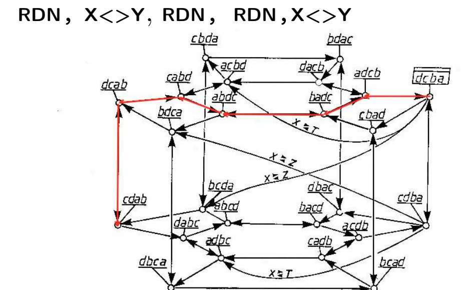
L'analyse combinatoire nous indique que le nombre de permutations possibles de 4 éléments est égal à 4! soit 24 états possibles.


Départ
dcba
cdba
bcad
adcb
cbad
bdac
acdb
dbac
cadb
dacb
badc
bacd
x ⇄ T
x ⇄ Z
x ⇄ Z
x ⇄ T
acbd
abdc
bcda
abcd
adbc
cbda
cabd
dabc
bdca
dbca
cdab
dcab

Le logigraphe est une représentation dans l'espace, sous la forme d'un graphe sagittal de cet ensemble de positions. La transition de l'une à l'autre s'effectue grâce aux instructions **X<>Y** (double flèche ↔), **RDN** (flèche simple ↓) ou **R↑** opération inverse de la précédente.

Il se présente sous la forme de deux ensembles hexagonaux superposés. Sur chaque hexagone se greffent trois cycles, chacun correspondant à une permutation circulaire complète des éléments de la pile (quatre **RDN** successifs). On passe d'un cycle à l'autre par la fonction **X<>Y** qui modifie l'ordre des valeurs.

L'utilisation du logigraphe est simple : on code la position finale désirée, soit "cdab" pour l'exemple qui précède, puis on repère cette combinaison sur le logigraphe, on cherche alors le plus court chemin qui relie cette combinaison à la position initiale. Toujours dans notre exemple on obtient la séquence **RDN, RDN, X<>Y, RDN, RDN,X<>Y**



Cette séquence d'instructions est celle que nous aurions utilisée si nous disposions d'une HP-65 ou 67. Mais `HP-41` dispose d'une instruction générale d'échange de registre qui nous permet d'obtenir deux nouvelles fonctions de permutation de registre de la pile : **X<>Z** et **X<>T.**

Si on reporte ces fonctions sur le logigraphe on voit que ces deux instructions créent de véritables court-circuit à travers la structure. En les utilisant la séquence précédente de six instructions se ramène à deux : **X<>Z,RDN**.

Pour écrire les programmes qui composent ce livre le logigraphe nous a apporté une aide irremplaçable et nous trouvons inconcevable de commencer la rédaction d'un programme sans en avoir une copie sur la table de travail.

À titre d'exemple de mouvements dans la pile nous vous proposons une première procédure : **TRIST** (TRI STACK, mélange affreux mais pas triste de français et d'anglais). Son but est d'*ordonner 4 valeurs contenues dans les registres de la pile*.

```
01•LBL "TRIST"
02 X>Y?
03 X<>Y
04 X<> T
05 X>Y?
06 X<>Y
07 X<> Z
08 X>Y?
09 X<>Y
10 R↑
11 X>Y?
12 X<>Y
13 X<> T
14 X>Y?
15 X<>Y
16 R↑
17 X>Y?
18 X<>Y
19 FC? 00
20 RTN
21 X<> Z
22 R↑
23 .END.
```

Si le drapeau 00 est désarmé cette procédure prend fin à la ligne 20. Si ce drapeau est armé les pas 21 et 22 ont pour effet d'inverser le contenu de la pile (voyez le logigraphe) ce qui fournit un classement par ordre croissant de X à T.

OPÉRATIONS ET REGISTRE L

Postulat : tout traitement arithmétique simple de quatre valeurs peut toujours être traité dans la pile.

Avant de développer ce principe fondamental et ses applications, nous devons préciser le rôle de chacun des registres de la pile opérationnelle.

Le registre X

C'est le point nodal, le lieu où tout converge. C'est l'ombilic de la pile. Bien que sur HP-41 ce registre ait perdu, grâce aux fonctions **VIEW** et **AVIEW**,

son rôle de fenêtre de communication étroite, il reste impliqué dans la totalité des opérations arithmétiques, logiques, ou de transfert de valeurs, effectuées par la machine. Les problèmes posés par l'utilisation de la pile se ramènent pour l'essentiel à placer dans le registre X la bonne valeur au bon moment.

Le registre Y

Il est un peu le faire valoir du précédent. Son rôle est plus restreint pour deux raisons :

– de nombreuses opérations ne demandent qu'un seul argument parmi les fonctions disponibles neuf d'entre elles seulement utilisent conjointement le registre Y et le registre X (**+, -, *, /, $\mathbf{Y}^x$, %, Σ+, Σ-**).
– l'extension des opérations d'arithmétique directe à tous les registres de la pile fait que le registre Y n'est plus obligatoirement l'opérande de ces opérations.

Le registre T

Il allie la puissance à la fragilité. Toute valeur contenue dans ce registre est finalement vouée à deux destins contraires soit son élimination pure et simple lors de l'introduction dans la pile d'une nouvelle valeur, soit sa duplication après une opération combinant X et Y.

À propos du registre T, nous ne pouvons résister au plaisir de vous présenter un ancêtre de l'utilisation de la pile opérationnelle qui sévissait déjà à l'époque lointaine où les calculateurs n'étaient pas programmables, voici l'algorithme de HORNER de résolution des polynômes :

$x^4 + 3x^3 - 6x^2 + 4x - 1$

Ceci est un polynôme de degré quatre qui a bien voulu nous servir d'exemple. Le problème est d'obtenir rapidement sa valeur pour un x donné.

Ce polynôme peut être réécrit sous la forme :

$(((x + 3) \times x) + (-6)) \times x + 561$

ce qui n'apparaît pas immédiatement comme une simplification, notons cependant que le calculateur n'effectue plus d'élévation de puissance mais

uniquement des additions et des multiplications, opérations beaucoup plus rapides. La notation polonaise inverse n'utilisant pas de parenthèses on peut espérer une écriture plus simple, enfin, suprême raffinement, les propriétés du registre T nous permettront de n'introduire qu'une fois la valeur de la variable qui sera reproduite à chaque opération.

L'algorithme devient finalement :

Saturer la pile (et en particulier T) avec la valeur de la variable X
(**X** ENTER↑ ENTER↑ ENTER↑)
RÉPÉTER Introduire le coefficient de rang N (par ordre décroissant) avec son signe
Additionner
Multiplier
JUSQU'À ce qu'il n'y ait plus de coefficients.
Ajouter la constante avec son signe.
FIN

Pour notre polynôme nous obtenons :

```
01 ENTER↑
02 ENTER↑
03 ENTER↑
04 3
05 +
06 *
07 6
08 CHS
09 +
10 *
11 4
12 +
13 *
14 1
15 -
16 .END.
```

C'est simple, rapide, élégant et toujours d'actualité ! Mais revenons à nos registres.

Le registre Z

Il a pour seule particularité de n'avoir rien de particulier. C'est par excellence le registre de stockage des résultats intermédiaires.



Le registre L

Fait-il vraiment partie de la pile ? Il semblerait que non. L'instruction **CLST** (CLEAR STACK) ne l'efface pas et, si vous disposez de l'imprimante il est ignoré par l'instruction **PRSTK** qui affiche le contenu de la pile.

Logiquement, cependant, la réponse est affirmative : Toutes les opérations, à l'exception de l'arithmétique directe, chargent dans L le contenu de X.

Ce registre reste cependant un îlot de stabilité dans l'univers tourbillonnant de la pile. Seules les instructions **LASTX** et **X<>L** permettent de réintroduire la valeur qu'il contient dans le carrousel. Il apparaît donc tout désigné pour jouer le rôle d'un accumulateur...

Après cette brève présentation des personnages, mettons les en scène sur un scénario classique.

RÉFLEXION AU SECOND DEGRÉ

Le manuel de l'utilisateur des pages 135 à 138 présente deux versions d'un programme de résolution d'équations du second degré. La seconde version, destinée à montrer l'utilité des sous-programmes, permet une économie de 7 lignes sur les 45 que compte la première.

La version que nous vous proposons effectue ce travail en 14 lignes (70% d'économie) et, avantage supplémentaire, aucun registre extérieur à la pile n'est utilisé.

Mais examinons d'abord la version proposée par HEWLETT-PACKARD : le sous-programme calcule le déterminant, $DELTA = b^2 - 4ac$, et, après un tronc commun, pour chaque racine on ajoute ou retranche ce déterminant à la valeur $-b$, le programme se termine par la traditionnelle division par $2a$. Comme à l'école !

Il n'est pas question de critiquer le classicisme de cette méthode, mais sans refaire les mathématiques on peut pousser un peu plus loin l'analyse pour réécrire la formule sous une forme plus agréable sinon pour l'élève moyen du moins pour le calculateur :

$$x = \frac{-b + \sqrt{b^2 - 4ac}}{2a}$$

Peut s'écrire également :

$$x = \frac{-b}{2a} + \sqrt{\frac{b^2}{4a^2} - \frac{c}{a}}$$

avec $E = {}^{-b}/_{2a}$ et $F = c/a$

$$x = E + \sqrt{E^2 - F}$$

Il n'y a donc que 2 expressions à calculer à partir des 3 valeurs a, b, c. Nous disposons de 5 registres dans la pile, c'est donc plus qu'il n'en faut.

Si à l'entrée de la procédure a est en X, b en Yet c en Z :

- ${}^{c}/_{a}$ s'obtient par **CHS**, **ST/Z**
- ${}^{b}/_{2a}$ s'obtient par **ST+X, /**

L'exécution de la fonction x^2 nous donne en X ${}^{b^2}/_{4a}$ et ${}^{-b}/_{2a}$ est sauvegardée en L. Il faut rappeler cette dernière valeur pour préparer le calcul des deux racines par **LASTX**, **STO T**, **RDN**.

Calculons pour finir notre pseudo-déterminant que l'on additionne et soustrait pour obtenir nos racines :

```
01♦LBL "CORNEIL"
02 CHS
03 ST/ Z
04 ST+ X
05 /
06 X↑2
07 LASTX
08 STO T
09 RDN
10 +
11 SQRT
12 ST- Z
13 +
14 .END.
```

Cet exemple vous aura permis d'apprécier la souplesse que confère l'arithmétique directe dans la pile. Il démontre également que le manuel n'est pas parole d'évangile mais constitue cependant une bonne base de départ pour développer votre curiosité.

LA CHASSE AU FORMAT

Si on associe maintenant les indicateurs binaires, le registre L , et l'arithmétique directe dans la pile, on peut obtenir de très jolies choses.

Imaginons qu'une procédure modifie le nombre de décimales affichées, nous désirons cependant restaurer l'affichage initial. Il est alors indispensable de connaître ce nombre de décimales avant de le modifier. Pouvons-nous arracher au calculateur ce renseignement précieux ?

Il existe ainsi que nous l'avons vu dans le chapitre II, mais codé sous forme binaire par les indicateurs 36 à 39. Rappelons que chaque drapeau indique une puissance de 2 et qu'il suffit d'additionner la valeur correspondante si l'indicateur est armé.

C'est ce que réalise la procédure **FM?** (FORMAT ?) en testant successivement ces quatre drapeaux. Cette procédure ne demande aucun paramètre d'entrée, le nombre de décimales est renvoyé dans X, les registres T et L sont perdus :

```
01♦LBL "FM?"
02 0
03 SIGN
04 FS? 39
05 ST+ L
06 ST+ X
07 FS? 38
08 ST+ L
09 ST+ X
10 FS? 37
11 ST+ L
12 ST+ X
13 FS? 36
14 ST+ L
15 X<> L
16 .END.
```

La fonction **SIGN** au pas 02 a deux buts : placer la valeur 0 dans le registre L qui va nous servir d'accumulateur et initialiser à 1 (2^0) le registre X. Des pas 04 à 14 le drapeau correspondant à la puissance de 2 présentée en X est testé et s'il est armé cette valeur est ajoutée au registre L. L'instruction **ST+X** double ce registre et permet de passer à la puissance de 2 suivante. En fin de programme le résultat final est replacé dans X par la fonction **X<>L**. Le contenu de L au retour de la procédure est donc toujours égal à 8.

Résumons les avantages apportés par l'arithmétique directe dans la pile :

– ces opérations ne perturbent pas l'ordre de la pile, un seul registre est modifié. (Le contenu du registre L est préservé)
– si l'opérateur est impérativement en X, l'opérande peut, lui, se situer dans l'un quelconque des registres de la pile (y compris en X comme nous venons de le voir).

Toujours dans le domaine de l'arithmétique classique voici un autre exemple de l'utilisation du registre L comme accumulateur : **le changement de base**.

L'homme a dix doigts, et comme il ne s'est pas abaissé à compter avec ses orteils la numération qu'il utilise est dite en base 10. En fait toutes les bases sont envisageables, un ordinateur comptera en base 2, et s'il existait des

composants électroniques à trois états stables on pourrait envisager d'utiliser la base trois.

La procédure suivante permet d'obtenir la représentation en base b d'un nombre en base 10, avec b compris entre 1 et 99.

Pendant l'exécution du programme, la pile contient : N nombre en base 10, b base d'arrivée, Nb le nombre en base b, 10^i où i est la puissance de b en cours de calcul.

Si b est supérieur à 10, chaque chiffre de la nouvelle base est exprimé sur deux chiffres.

```
01*LBL "10-b"      Initialisation de Nb et 1Ø i.
02 1
03 0
04 RDN
05 RDN
06*LBL 01
07 X<>Y
08 /
09 LASTX
10 X<>Y
11 INT
12 ST- L
13 X<>Y            Reste de la division en L
14 ST* L
15 R↑
16 ST* L
17 X<> L
18 ST+ T           Addition du nouveau chiffre dans T.
19 CLX
20 10
21 X<Y?
22 ST* X           Multiplication par 1Ø ou par 1ØØ en fonction de
23 ST* L           la valeur de b.
24 X<> L
25 RDN
26 X<>Y
27 X≠0?
28 GTO 01
29 X<> Z
30 .END.
```

Pour utiliser cette fonction il faut introduire la base puis le nombre et terminer avec **XEQ "10-b"**. Le résultat est alors présenté en X. Les contenus des registres Z, T et L sont perdus. La base *b* est conservée en Y.

Exemple :

2 ENTER↑ 255 →11111111

16 ENTER↑ 255 →1515

Dans ce deuxième exemple la base étant supérieure à 10 chaque groupe de deux chiffres représente un seul chiffre dans la base d'arrivée, le résultat est donc 15 15 soit FF dans la notation hexadécimale.

À titre d'entraînement, pour vous échauffer les neurones, voici un petit exercice :

Soit la suite :

10 - 11 - 12 - 13 - 14 - 15 - 20 - XX - 30

Quelle peut être la valeur représentée par **XX** ?

À LA "MOD" DE CHEZ NOUS

Après cette étude des quatre opérations arithmétiques nous allons nous intéresser à des fonctions plus complexes dont HP-41 est abondamment pourvu. L'une d'elle, la fonction **MOD** (MODULO), avec laquelle nous allons explorer le riche domaine du calcul fractionnaire.

Sous ce mystérieux nom de modulo se cache une opération connue de tous les écoliers : le reste de la division entière.

Souvenez-vous :

En 7 combien de fois trois, il y va 2 fois. Je pose 2. 2 fois 3 égale 6, ôté de 7 il reste 1.

On peut utiliser un langage plus évolué et dire que *7 est congrue à 1 modulo 3.*

Enfin si vous introduisez dans le calculateur la séquence : **7 ENTER↑ 3 XEQ "MOD"**, le résultat sera 1.

Écolier, mathématicien, calculateur ont tous trois fait la même opération.

Pour mieux saisir l'utilité de cette fonction nous allons développer un logiciel de traitement des fractions. Notre point de départ a été le programme "*PPCM, PGCD, et approximation fractionnaire d'un nombre décimal*", présenté dans la bibliothèque mathématique, destinée aux calculatrices HP-67 et 97 et très exactement localisé des pages 12 à 15.

Ce programme calcule :

- le Plus Grand Commun Diviseur, PGCD ;
- le Plus Petit Commun Multiple, PPCM ;
- la forme réduite d'une fraction ;
- les valeurs s et t solutions de l'équation : $a \times s + b \times t = pgcd(a, b)$;
- la fraction entière approchant au mieux un nombre décimal donné.

Pour effectuer toutes ces opérations le programme fourni par HEWLETT-PACKARD utilise 199 instructions et 7 registres de données. Sur HP-41 nous effectuerons les mêmes fonctions en 86 instructions sans déborder de la pile opérationnelle.

Envisageons d'abord le calcul du PGCD. L'algorithme utilisé est celui d'EUCLIDE, célèbre géométricien grec qui, tout en postulant, poursuivit parallèlement des recherches en arithmétique.

Le principe est le suivant : on remplace le dénominateur par le reste de la division entière (le modulo) et le numérateur par le dénominateur. Si le nouveau dénominateur est nul le calcul s'arrête et le numérateur est alors égal au PGCD. Sinon on réitère l'opération sur le nouveau rapport. Il est aisé d'écrire le sous-programme de 6 lignes.

```
01*LBL 00        Voici la vedette
02 MOD
03 LASTX
04 X<>Y
05 X≠0?
06 GTO 00
07 .END.
```

Le couple de valeurs initiales a et b est introduit en X et Y, dans un ordre quelconque. En fin d'exécution le registre X contient 0 et le registre Y le PGCD. On ne peut pas faire plus concis !

Le PGCD est habituellement utilisé pour obtenir la forme réduite d'une fraction (on dit aussi simplifier).

Il est donc important et opportun de modifier légèrement notre séquence pour obtenir la fraction a'/b' réduite en divisant les deux termes de la fraction par leur PGCD. Pour parvenir à ce résultat il nous faut d'abord sauvegarder les valeurs a et b dans les registres Z et T. On obtient finalement :

```
01♦LBL "RF"     Réduction de Fraction.
02 STO Z        Sauvegarde du dénominateur.
03 X<>Y
04 STO T        Sauvegarde du numérateur.
05♦LBL 02
06 MOD          Calcul du PGCD.
07 LASTX
08 X<>Y
09 X≠0?
10 GTO 02
11 +            Rappel du PGCD en X.
12 ST/ Z
13 ST/ Y        Obtention de la fraction réduite
14 RDN
15 .END.
```

Sauriez-vous retrouver l'unique instruction à ajouter à cette séquence pour calculer le Plus Petit Commun Multiple ?

Rappelions que ce dernier est égal au rapport du produit des deux nombres par leur PGCD, ou encore au produit du numérateur de la fraction (a) par le dénominateur de la fraction réduite (b'). La valeur b' est actuellement en X et le numérateur a est présent en Z, il a en effet été recopié par T lors de l'addition effectuée au pas 11.

L'instruction à ajouter est donc simplement :

```
15 ST* Z
```

Suivi de **16 END** pour clore la procédure.

Voyons sur un exemple le fonctionnement de l'ensemble : soit la fraction $91/65$, calculer sa forme réduite, le PPCM et le PGCD de ses deux termes :

91 [ENTER↑] 65 XEQ "RF" → 5 dénominateur fraction réduite.

RND → 7 Numérateur fraction réduite

RND → 455 PPCM

RND → 13 PGCD

L'heure est venue de nous préoccuper du calcul de s et t. Comme pour la résolution des équations du second degré, il faudra reconsidérer l'algorithme classique et proposer une autre méthode, peut-être moins immédiate pour le calculateur humain, mais mieux adaptée à la machine.

L'équation de départ : $a \times s + b \times t = PGCD(a, b)$ peut être réécrite en divisant les deux membres par le PGCD.

On obtient a' les termes de la fraction réduite, que nous savons maintenant calculer. Nous allons donner successivement à t les valeurs entières 0, 1, -1, 2, -2… et calculer $s = \frac{(1-b' \times t)}{a'}$ jusqu'à ce que s prenne une valeur entière.

```
01♦LBL "ST"
02 XEQ "RF"      Calcul de la fraction réduite
03 0
04 1             Initialisation de T.
05♦LBL 03        Début de la boucle.
06 RDN
07 CHS
08 X<=0?         Décrémentation de 1 si négatif.
09 DSE X
10 STO X         Inhibe le saut après DSE.
11 STO T
12 X<>Y
13 *
14 LASTX
15 RDN
16 CHS
17 ISG X         1-b'*t,
18 STO X         Inhibe le saut après ISG
19 X<>Y
20 /             Obtention de s en X.
21 LASTX
22 RDN           Récupération de a'.
23 FRC
24 X≠0?          Test : s est-il fractionnaire ? Si oui on itère.
25 GTO 03
26 X<> L
27 .END.
```

En fin de calcul s est en X et t en Y.

Exemple : toujours avec les valeurs 91 et 65 :

91 **ENTER↑** **65** **XEQ "ST"** → 2 valeur de S

X<>Y → 3 valeur de T

Nous obtenons bien :

$91 \times (-2) + 65 \times 3 = 13$: PGCD (91, 65)

Dans la suite de nos travaux, inspirés de ceux d'HERCULE, nous vous avons annoncé le calcul de la fraction approchant au mieux, compte-tenu de la précision du calculateur, un nombre décimal donné.

L'algorithme employé dans ce programme utilise une méthode de fraction continue. Ne nous demandez pas de vous l'expliquer en détail, nous ne sommes pas certains d'avoir nous-mêmes tout compris !

Le programme, par contre, fonctionne fort bien :

```
01◆LBL "DF"
02 0
03 1
04 RCL Z
05 FIX 0
06◆LBL 04
07 FRC
08 1/X
09 INT
10 X<>Y
11 ST* Y
12 RDN
13 X<>Y
14 ST+ Y
15 X<> L
16 RDN
17 ST* Y
18 X<>Y
19 RND
20 X<> L
21 X<>Y
22 CLA
23 ARCL L
24 "├/"
25 ARCL X
26 AVIEW
27 ST/ Y
28 ST/ L
29 X<> L
30 X=Y?
31 GTO 00
32 X<> L
33 X<>Y
34 X<> T
35 GTO 04
36◆LBL 00
37 LASTX
38 ST* Y
39 .END.
```

Les instructions 21 à 25 utilisent les possibilités alphanumériques du calculateur pour afficher clairement à chaque approximation la fraction obtenue. Si les valeurs affichées défilent trop vite à vos yeux vous pouvez armer manuellement le drapeau 21, le calculateur s'arrêtera alors à chaque itération.

Nous donnerons comme exemple les approximations fractionnaires du nombre π

SF 21 π XEQ "DF" → 22/7

R/S → 333/106

R/S → 355/113

R/S → 104348/33215

R/S → 33215

La dernière valeur affichée est le dénominateur de la dernière fraction obtenue par HP-41 le numérateur est en Y. Cette approximation est la meilleure que peut fournir le calculateur du fait de sa connaissance limitée des décimales de π.

Il a fallu près de 15 siècles pour passer de la première approximation qui est celle d'ARCHIMÈDE à la dernière calculée par votre HP-41 en 6 secondes !

Nous avons rempli notre contrat mais nous ne pouvons résister à la tentation d'en rajouter un peu. Voici sans commentaire (en est-il besoin ?) un groupe de procédures très brèves qui vous permettront d'effectuer additions, soustractions, multiplications et divisions sur des fractions. La fraction opérande est à l'entrée en T et Z, la fraction opérateur en Y et X. Le résultat final est également en Y et X.

```
01♦LBL "F/"
02 X<>Y
03♦LBL "F*"
04 ST* Z
05 RDN
06 ST* Z
07 RDN
08 GTO "RF"
09♦LBL "F-"
10 CHS
11♦LBL "F+"
12 ST* T
13 X<> Z
14 ST* Z
15 *
16 RCL Z
17 +
18 X<>Y
19 GTO "RF"
20 .END.
```

Problème : quelle résistance mettre en parallèle avec une autre de 100 ohms pour obtenir une valeur finale de 80 ohms ?

On sait que $1/R' + 1/R" = 1/Rt$ et donc
$1/R" = 1/Rt - 1/R'$

Il vous suffit d'exécuter :

1 **[ENTER↑]** **80**
[ENTER↑] **1** **[ENTER↑]** **100** (deuxième fraction)

XEQ "F-" → 400 valeur recherchée.

Vous pouvez vous assurer au passage que le numérateur est bien égal à 1, la fraction étant réduite à la suite de l'appel de **RF**.

La fonction modulo semble donc fort pratique, cependant, dans certains cas elle s'avère insuffisante. Il est parfois souhaitable d'obtenir non seulement le reste de la division mais aussi son quotient. Il est dit dans le manuel (pp. 11) : «*Si vous avez besoin d'une fonction qui n'est pas micro-programmée, vous pouvez écrire un programme répondant à ce besoin.* »

Écrivons donc la procédure **DIV** (DIVISION ENTIÈRE) en lui imposant les spécifications suivantes :

– entrée : dividende en Y, diviseur en X ;
– sortie : quotient en X, reste en Y, le diviseur est sauvegardé en L ;
– conservation des registres Z et T.

Pourrons-nous respecter ces contraintes drastiques ? Vous le saurez en deux épisodes :

Premier épisode :

```
01♦LBL"DIV"
02 ST/ Y
03 X<>Y
04 INT
05 ST- L
06 X<>Y
07 ST* L
08 X<> L
09 X<>Y
10 .END.
```

Voilà une procédure brève et élégante, vérifions qu'elle donne de bons résultats :

13 **ENTER↑** **3 XEQ "DIV"** → **4** le quotient

X<>Y → **1** le reste

LASTx → **3** le reste

Êtes-vous satisfait ? N'y aurait-il pas un léger défaut ? Cherchez-le, nous vous donnons rendez-vous dans quelques instants pour en discuter.

Deuxième épisode :

Hé oui ! vous avez bien vu, quand on reprend l'exemple qui précède en affichant 9 décimales on découvre l'horrible vérité : le reste n'est pas égal à 1 mais à 0.999999999. Une erreur d'un milliardième peut avoir des conséquences dramatiques surtout si HP-41 compte sur ses décimales pour effectuer des opérations spécifiques (voir au chapitre suivant le problème des codes de contrôle).

Il nous faut donc rédiger une nouvelle version, moins brève, moins élégante, mais plus exacte. Pour éviter les décimales erronées nous sommes contraints d'effectuer un arrondi en remplaçant le pas 08 du programme précédent par :

```
08 CLX
09 .5
10 ST+ L
11 X<> L
12 INT
13 .END.
```

Voilà un programme bien rafiscotché : nous renonçons au passage à conserver le diviseur dans le registre L. En contrepartie cette procédure calcule un résultat toujours exact. C'est néanmoins avec tristesse que nous renonçons aux spécifications initiales, si vous trouvez une solution satisfaisante à ce problème, n'hésitez pas à nous en faire part.

Cette histoire a une morale : quand vous effectuez des calculs arithmétiques placez-vous toujours en FIX 9 pour vérifier si par hasard le calculateur ne vous présenterait pas, sous un emballage irréprochable, une marchandise frelatée.

QUELLES COMBINAISONS !

Une autre fonction originale de HP-41 est la fonction factorielle (**FACT**).

Factorielle de n, (notée aussi $n!$) est le résultat du produit des n premiers entiers.

Nous connaissons tous l'histoire du sage de l'Inde qui demanda comme salaire un grain de blé sur la première case d'un échiquier, deux grains de blé sur la seconde, quatre sur la suivante et ainsi de suite en doublant le nombre de grains sur chaque case jusqu'à la soixante quatrième. L'opération n'était pas mauvaise mais il aurait pu s'enrichir encore plus vite en dédaignant l'exponentielle pour lui préférer la factorielle.

En effet, si 2^{64} est égal à $1,8E19$, $64!$ vaut $1,8E89$ (c'est probablement ce genre de résultat qui justifie le point d'exclamation comme symbole de l'opération factorielle).

Cette opération est à la base de l'analyse combinatoire dont le but est de dénombrer les états que peuvent prendre un nombre fini d'objets. $n!$ est égal au nombre de permutations distinctes de n objets.

Pour connaître le nombre de configurations de la pile opérationnelle à quatre registres il nous a suffi de faire :

4 XEQ "FACT" → 24

Si on prend en compte un cinquième registre (L) le nombre de permutations est alors de 120. Nous vous laissons le soin de tracer le logigraphe correspondant.

Une autre valeur intéressante est le nombre de combinaisons de k objets pris parmi n. On peut ainsi calculer le nombre de mains qu'il est possible d'obtenir au jeu de bridge sachant que chaque main est constituée de 13 cartes tirées d'un paquet de 52.

La formule qui permet d'effectuer ce calcul est la suivante :

$$\binom{n}{k} = \frac{n!}{k!(n-k)!}$$

C'est assez barbare mais cela se programme fort bien :

```
01♦LBL "CB"
02 X<>Y
03 FACT
04 X<>Y
05 ST- L
06 LASTX
07 FACT
08 X<>Y
09 FACT
10 *
11 /
12 .END.
```

Nous pouvons maintenant résoudre le problème posé plus haut :

52 ENTER↑ 13 XEQ "CB" → 6,35E11

Si vous êtes joueur de bridge traitez avec respect les mains qui vous échoient, elles représentent la réalisation d'un événement qui n'avait qu'une chance sur 635 milliards de se produire, et donc potentiellement. unique.

Pour les amateurs de curiosités mathématiques voici deux charmants problèmes dont la résolution nécessitera l'utilisation de **CB** :

- Le piquet est un jeu de cartes où la seule annonce autorisée est "Dix de blanc". Cela signifie que parmi les douze cartes qui lui ont été distribuées le joueur ne possède aucune figure. Quelle est la probabilité de faire cette annonce sachant que le piquet se joue avec 32 cartes ?
- Lorsqu'une équation a de nombreuses solutions évidentes il peut être plus passionnant de déterminer leur nombre que leurs valeurs. Combien l'équation : $x + y + z + t = 13$ a-t-elle de solutions entières, strictement positives ?

Puisque nous avons abordé le terrain des probabilités terminons avec une ultime application de la procédure **CB**.

La l*oi binomale* permet de calculer la probabilité qu'un événement favo-

rable se réalise k fois sur n épreuves indépendantes en fonction de la probabilité p d'apparition de l'évènement pour chaque épreuve.

Dans une famille de six enfants combien a-t-on de chances de n'avoir que des filles ? La réponse est donnée par la formule :

$$p = \begin{pmatrix} n \\ k \end{pmatrix} \times (1-p)^{n-k}$$

Dans le programme **LBN** (Loi BiNomiale) nous calculerons d'abord les termes exponentiels avant l'appel de **CB**.

```
01*LBL "LBN"
02 1
03 X<>Y
04 -
05 LASTX
06 X<> Z
07 ST- T
08 X<> T
09 Y↑X
10 LASTX
11 X<> Z
12 R↑
13 ST+ T
14 Y↑X
15 LASTX
16 RDN
17 *
18 STO T
19 RDN
20 X<>Y
21 XEQ "CB"
22 *
23 .END.
```

Spécifications :

- Entrée n en Z, k en Y, p en X ;
- sortie p en X ;
- Les registres T et L sont perdus.

Nous pouvons maintenant résoudre nos problèmes familiaux :

6 ENTER↑ (nombre d'épreuves, ici le nombre d'enfants, sic)

6 ENTER↑ (nombre de fois où l'évènement se réalise)

0,5 (Probabilité d'avoir une fille)

XEQ "LBN" → **0,0156** soit une chance sur 64.

En pratique, le calcul qui précède est probablement faux : la probabilité d'avoir une fille est un peu supérieure à 1/2 (environ 0,52) et surtout, il n'est sûr du tout que les "tirages" successifs soient indépendants !

CONCLUSION :

Nous espérons que la description des nombreuses procédures qui précèdent vous ont permis de découvrir les possibilités très étendues de la pile opérationnelle. Nous aurions pu multiplier les exemples à l'infini, mais l'épaisseur d'un livre est nécessairement limitée. Nul doute qu'une de ces procédures vous sera utile un jour, et si, malgré tout, vous n'y trouvez pas votre bonheur écrivez donc vos propres fonctions pour mieux accroître la puissance et le confort d'utilisation de votre `HP-41` .

CHAPITRE IV

HASARD NÉCESSAIRE



Nous nous sommes longtemps torturé l'esprit pour savoir dans quel chapitre placer les deux procédures indispensables qui suivent. La solution a soudain jailli de nos têtes surmenées : il suffisait de créer un chapitre supplémentaire, le voici :

Nous traiterons ici du hasard. Le hasard est partout il gouverne nos vies. Aussi bien dans nos jeux (Loto, Tiercé) que dans des activités plus sérieuses comme la recherche scientifique, voire la politique.

Pour savoir maîtriser le hasard il faut pouvoir le simuler. Comment y parvenir, en utilisant un outil aussi fortement déterministe qu'un ordinateur ?

Le hasard selon certains n'est que la mesure de notre ignorance, il nous suffira de demander au calculateur d'effectuer une suite d'opérations suffisamment complexe pour que le résultat nous paraisse "imprévisible".

Cela ne suffit pas, si la valeur initialement fournie au calculateur est toujours la même, le calcul, aussi compliqué soit-il, fournira toujours les mêmes résultats. Il faut donc initialiser le processus aléatoire. Comme le calculateur est incapable de le faire c'est à l'homme que reviendra la tâche d'accomplir le premier pas sur le chemin du hasard.

Des deux procédures annoncées, la première **HSD** (HASARD) nous permettra d'introduire une "semence" dans un registre du calculateur, et la seconde **GAG** (GÉNÉRATEUR ALÉATOIRE GÉNÉRAL) utilisera et modifiera cette valeur au cours des tirages successifs.

HSD teste si le registre R00 contient un nombre à partie fractionnaire non nul. Si le test est vérifié, **HSD** se branche sur **GAG** histoire de brouiller un peu les pistes. Dans le cas contraire (nombre entier ou chaîne de caractères dans R00) le calculateur s'interrompt pour vous demander une semence. La validité de cette semence est vérifiée par un retour en début de programme.

```
01♦LBL"HSD"
02 DEG
03 RCL 00
04 SF 25
05 FRC
06 FS?C 25
07 X=0?
08 FS? 30
09 GTO 00
10 RDN
11 "ALEA 0<X<1 ?"
12 TONE 9
13 PROMPT
14 STO 00
15 RDN
16 GTO "HSD"
17♦LBL 00
18 CLX
19♦LBL"GAG"
20 FRC
21 ST- L
22 1 E3
23 ST* Y
24 X<> L
25 X>Y?
26 X<>Y
27 -
28 ISG X
29 STO X
30 RCL 00
31 ST* Y
32 X<> L
33 +
34 INT
35 RCL 00
36 ACOS
37 FRC
38 STO 00
39 RDN
40 .END.
```

La procédure **GAG** demande un paramètre d'entrée un peu particulier : c'est un nombre qui en regroupe deux. L'un occupe la partie entière, l'autre les trois premiers chiffres de la partie fractionnaire, on peut représenter la chose sous le format *nnn*, *mmm* (ce que nous appellerons un code de contrôle).

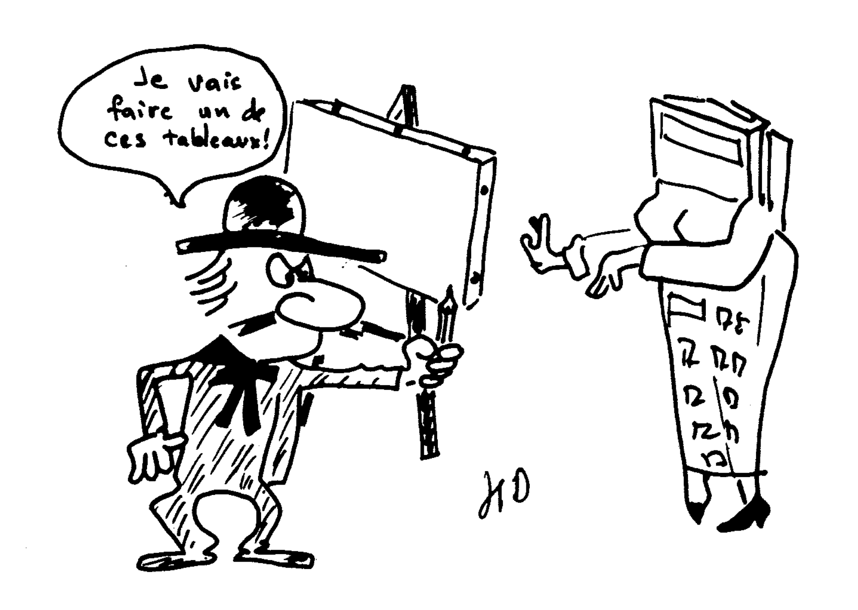
À partir de ce code, **GAG** renvoie un nombre entier aléatoire, compris entre *nnn* et *mmm* bornes incluses, en, X. Un nombre aléatoire compris, lui, entre 0 et 1 bornes exclues, est également retourné en T. Les registres Y et Z sont respectés, L est perdu.

Mode d'emploi de l'ensemble : dès les premiers pas d'un programme utilisant des nombres aléatoires, on appelle **HSD**. Si cette procédure estime que le contenu de R00 peut être utilisé comme semence il n'y a pas d'interruption du programme.

Quand on désire obtenir un nombre aléatoire entre *nnn* et *mmm*, on crée le code *nnn*, *mmm* suivi de **XEQ "GAG"**. Précisons que l'ordre des valeurs *nnn* et *mmm* est indifférent. Ainsi les codes ,003 et 3 renverrons tous deux un nombre parmi 0, 1, 2 ou 3. La deuxième forme est donc pour des raisons d'encombrement nettement préférable.

Mais il est temps de commencer la lecture du chapitre suivant dans lequel vous trouverez précisément une procédure **MEL** lui utilise **GAG** ; est-ce un hasard ?...

CHAPITRE V
SUR TOUS LES TABLEAUX



Parmi les variables traitées en informatique certaines sont solitaires : le rayon d'un cercle, le rapport d'un tiercé ou l'omniprésent âge du capitaine.

Mais, supposons que nous nous intéressions subitement à plusieurs capitaines, à une statistique sur les rapports du tiercé pendant un an, ou à une famille de cercles. Ce n'est plus une seule valeur qui retient notre attention mais un ensemble. Nous chercherons à manipuler cet ensemble comme un tout ou à découvrir certaines propriétés qui lient ces valeurs entre elles.

Cette suite de nombres ou de chaînes de caractères, pour des raisons de commodité, sera située dans un groupe de registres consécutifs.

Pour le **mathématicien** un tel ensemble prend le nom de vecteur. L'**informaticien** parlera, lui, plus volontiers de tableau.

Au cours de ce chapitre nous préciserons comment traiter ces groupes de valeurs, et définir une série de procédures générales de traitement.

S'EFFACER

Pour illustrer ces notions, débutons par un exemple simple : la remise à zéro d'un ensemble de mémoires. Le HP-41 dispose de quatre fonctions d'effacement :

- CLX pour effacer le registre X ;
- CLST pour effacer la pile opérationnelle ;
- CLΣ pour effacer les registres statistiques ;
- CLRG pour remettre à zéro la totalité des mémoires.

Mais aucune de ces fonctions ne permet d'effacer électivement un groupe de registres défini par l'utilisateur.

Plaçons-nous dans un cas particulier : nous désirons effacer les registres de R11 à R15. Il est possible d'écrire le programme :

```
01◆LBL "TCL1"
02 0
03 STO 11
04 STO 12
05 STO 13
06 STO 14
07 STO 15
08 .END.
```

Spécification de* TCL1 *:

– *données initiales* : aucunes ;
– *données finales* : aucunes ;
– *modifications* : perte du registre T.

Un tel programme remplit bien la tâche que nous nous sommes assignée. Il ne saurait cependant être considéré comme une procédure générale. En effet si les cinq registres à effacer ne sont plus situés entre R11 et R15, i l sera nécessaire de réécrire un nouveau programme.

Envisageons maintenant le cas, encore plus particulier, de l'effacement d'un unique registre : si l'emplacement de ce registre est connu au moment de l'écriture du programme, la solution est immédiate. Mais si l'adresse du registre doit être calculée, s'il s'agit d'une variable, il n'est pas possible de l'indiquer directement dans le programme.

On peut cependant déposer cette adresse calculée dans un registre dont l'adresse sera directement précisée à l'ordinateur. Ce dernier devra aller chercher à cette adresse l'adresse finale. Pour cela, il est nécessaire d'intercaler l'ordre **IND** (indirect) pour mettre le calculateur au courant de ce tour de passe-passe. Remarque : cette technique est d'ailleurs la seule qui autorise l'accès aux registres de rang supérieur à 99.

Ainsi, après avoir placé dans X le numéro d'un registre, la séquence

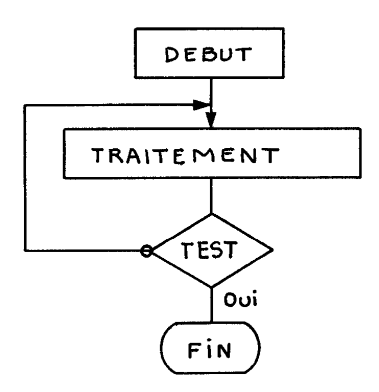
```
01 0
02 STO IND Y
```

effacera le contenu de ce registre.

Nous avons progressé vers la solution de notre problème puisque nous sommes maintenant capable d'atteindre, par une instruction, n'importe quel point de l'espace mémoire.

Pour accéder successivement à plusieurs points nous pourrions répéter plusieurs fois la séquence précédente en faisant varier le contenu de Y. Il est de loin préférable de faire appel à un autre outil puissant : `les boucles.`

Sur HP-41 les boucles ont généralement la structure suivante :



Les tenants de la programmation structurée diront, fort doctement, que cette boucle est du type FAIRE JUSQU'À... Il existe un autre type de boucle : TANT QUE... FAIRE, que nous envisagerons par la suite.

Revenons à notre procédure de nettoyage. Elle a maintenant besoin, pour s'exécuter, de deux paramètres : l'adresse du premier registre à effacer (la base du bloc) et celle du dernier (le sommet).

Nous pouvons écrire la deuxième version de **TCL**

```
01*LBL"TCL2"
02 0
03 STO IND Y
04 SIGN
05 +
06 X<=Y?
07 GTO"TCL2"
08 .END.
```

Le numéro du registre initial est en X, et celui du dernier registre à effacer en Y.

***Spécifications de* TCL2**

– *données initiales* : adresse de la base du tableau en X, adresse du sommet en Y ;
– *données finales* : aucune ;

– *modifications* : perte du registre T, la valeur finale en X est supérieure d'une unité à celle de Y inchangée.

Le pas 03 utilise l'adressage indirect pour effacer le registre dont l'adresse est présente dans Y.

Les pas 05 et 06 augmentent d'une unité cette adresse (on parle également de pointeur pour désigner cette valeur que l'on incrémente), la fonction **SIGN** est ici équivalente à deux instructions : **CLX**, **1**, elle est plus rapide et utilise un octet de moins.

Enfin, le pas 06 vérifie que l'adresse finale n'est pas atteinte. On retrouve bien la structure de boucle (FAIRE. . .JUSQU'À) telle que nous venons de la schématiser.

Cette procédure est parfaitement satisfaisante : quelle que soit l'adresse initiale et le nombre de registres à effacer elle pourra fonctionner. En fait, elle n'utilise pas à fond toutes les caractéristiques du `HP-41`. L'opération d'incrémentation suivie d'un test est à ce point courante qu'une instruction spéciale a été prévue Incrémentation et Saut si plus Grand : **ISG**.

Nous avons, dans **TCL2**, utilisé deux registres pour ranger initialement les deux paramètres de contrôle : les adresses de début et de fin.

L'instruction **ISG** regroupe ces deux données en un seul registre :

– l'adresse de la base constitue la partie entière ;
– l'adresse du sommet, dont la valeur ne peut excéder 3 chiffres, est située dans la zone fractionnaire.

L'ensemble forme un code : $ddddd, fff$ qui délimite ainsi une zone de la mémoire des données.

Cette notation est générale, elle est également utilisée par les périphériques, mémoires de masse et imprimante, aussi l'adopterons-nous. Désormais, nous désignerons un tableau en mémoire par ce paramètre : `le code de contrôle`.

Il ne nous reste plus qu'à utiliser ces nouvelles normes pour écrire une troisième (et dernière) version de **TCL** :

```
01*LBL"TCL"
02 SIGN
03 CLX
04*LBL 01
05 STO IND L
06 ISG L
07 GTO 01
08 .END.
```

Cette fois c'est le paramêtre ddd,fff qui est initialement en X.

Spécification **TCL** :

- *données initiales* : code de contrôle en X ;
- *données finales* : aucunes ;
- *modifications* : le registre X est perdu, le registre X prend finalement la valeur 0.

Ce dernier programme comporte également l'instruction **SIGN**, mais son rôle est maintenant de placer notre code de boucle dans le registre L. Le but de la manœuvre étant de perturber la pile opérationnelle le moins possible. Effectivement, une fois **TCL** exécutée, les registres Y, Z et T sont restés inchangés.

On aurait pu, pour parvenir au même résultat, utiliser au pas 02 l'instruction **STO L** plus naturelle, mais rappelons nous notre philosophie générale : une procédure doit être optimisée au maximum, elle doit être brève et former un bloc sur lequel il n'y aura plus à revenir. C'est une nouvelle fonction du langage de votre HP-41 , au même titre que **LOG** ou **PROMPT**, vous ne savez pas (et nous non plus) comment ces fonctions ont été programmées, vous avez également le droit d'oublier quelle suite d'instructions compose votre procédure. Par contre, il est impératif d'en noter soigneusement les conditions d'utilisation : paramètres d'entrée, résultats, modification des registres de la pile aussi bien que de la mémoire donnée, altération de l'état des indicateurs binaires. . .

Nous avons maintenant entre nos mains tous les outils nécessaires et suffisants à la création d'un ensemble homogène de procédures de manipulations des tableaux. Nous vous laissons le soin d'ouvrir le feu !

EN RAJOUTER

Pour bien vérifier que vous avez saisi, écrivez une procédure qui fait la somme de tous les éléments d'un tableau. Pour pimenter la chose cette procédure doit conserver les valeurs présentes en Y et en Z. Le paramètre de boucle *ddd*, *fff* est à l'entrée dans X, en fin d'exécution la somme obtenue est renvoyée dans ce même registre. Travaillez bien avant de regarder la suite !

```
01*LBL"TSM"
02 SIGN
03 CLX
04*LBL 02
05 RCL IND L
06 ST+Y
07 RDN
08 ISG L
09 GTO 02
10 .END.
```

Le programme **TSM** ressemble à **TCL** comme un frère, seule la partie traitement est modifiée (c'est un minimum) : le pas **05 STO IND L** est remplacé par 3 instructions. On aurait pu envisager d'écrire simplement + à la place de **ST + Y**, **RDN**. En fait, cela conduit droit à la catastrophe !

Souvenons nous que notre précieux paramètre de boucle est dans L, toute opération effectuée sur le registre X va de la façon la plus implacable écraser le contenu de L. La suite, s'il y en a une, est inracontable. Heureusement l'arithmétique directe dans les registres, étendue à la pile opérationnelle, permet de faire l'opération sans modifier le contenu de L.

Comme application de cette procédure nous vous proposons la brève séquence d'instructions qui suit. Elle nous permettra d'introduire en mémoire, sous forme de tableau, la suite des n premiers nombres entiers**05 STO IND L**.

```
01*LBL"SUITE
02 1 E3
03 /
04 1
05 +
06 SIGN
07*LBL 00
08 LAST X
09 INT
10 STO IND X
11 ISG L
12 GTO 00
13 .END.
```

Si vous exécutez cette procédure avec le nombre 10 initialement en X vous obtiendrez *in fine* : l dans le registre R01, 2 dans R02, etc... jusqu'à l0. Le code de contrôle de ce bloc de valeur sera $1,010$.

Introduisez maintenant ce code de contrôle en X puis exécutez **ISN** : vous obtenez 55. D'autres essais sur différents nombres de valeurs vous convaincront que la somme des n premiers nombres entiers est donnée par la formule :

$s(n) = \frac{n \times (n+1)}{2}$



UN MAXIMUM !

La partie traitement qui occupe le centre de nos boucles peut être aussi diverse qu'il est possible de l'imaginer.

Nous venons d'effectuer des stockages (**TCL** et **SUITE**), des opérations arithmétiques (**TSM**), voici maintenant un exemple qui utilise un test.

TMX recherche dans un tableau la plus grande des valeurs et renvoie son adresse. En cours de fonctionnement la pile doit contenir :

– le code de contrôle de la boucle ;
– la valeur maximum actuelle ;
– l'adresse de cette valeur ;
– l'élément du tableau que l'on est en train de comparer.

Tout cela fait beaucoup, mais pas trop. Il nous reste même un registre pour conserver quelque trace de l'ancien contenu de la pile. Pour pouvoir nous en sortir il nous faudra cependant tirer le maximum des propriétés de la pile, cela n'est plus pour vous effrayer (puisque vous avez lu le chapitre précédent).

Voici le programme :

```
01*LBL"TMX"
02 X<>Y
03 SIGN
04 RDN
05 RCL X
06 RCL IND X
07 ENTER↑
08*LBL 03
09 CLX
10 RCL IND Z
11 X<=Y?
12 GTO 00
13 X<>Y
14 LASTX
15 +
16*LBL 00
17 ISG Z
18 GTO 03
19 X<> L
20 R↑
21 .END.
```

Le code de contrôle ddd,fff est bien entendu en X.

Ce programme est déjà un peu plus long et plus complexe que les précédents. Découpons-le en tranches.

Les pas 02 à 04 (**X**, **RCL Y**, **SIGN**, **RDN**) ne servent qu'à sauvegarder le contenu de Y dans L, nous espérons l'y retrouver à la fin des opérations. Puis des pas 05 à 07 (**RCL X**, **RCL IND X**, **ENTER↑**) la pile est manipulée pour avoir la configuration suivante :

- T = pointeur du maximum courant ;
- Z = code de boucle (pointe sur la valeur à comparer) ;
- Y = valeur maximum courante ;
- X = valeur à comparer.

Au début de la procédure la base du tableau est utilisée comme valeur maximum.

La boucle de traitement commence au pas 08. Au pas 09 (**CLX**) le registre X est effacé, c'est une opération de nettoyage, et immédiatement une nouvelle valeur à comparer au maximum est rappelée (**RCL IND Z**).

Un test est alors effectué :

- si ce nouveau nombre est inférieur au maximum actuellement en Y, on achève ce tour de boucle sans rien modifier (branchement à l'étiquette 02) ;
- dans le cas contraire, nous avons deux choses à faire :

  - remplacer le maximum précédent, maintenant déchu, par son challenger, un simple **X ↔Y** suffit,
  - puis remplacer le pointeur dans T par l'adresse de notre nouveau maximum qui se trouve actuellement en Y.

- Ceci pourrait être obtenu par des mouvements de la pile du style : **R↑**, **CLX**, **RCL T**, **RDN**. Il est cependant possible de faire plus court. Les pas 14 et 15 (**LASTx**, +) suffisent car ils utilisent les propriétés des registres T et L. Le rappel de **LASTx** éjecte de la pile l'ancien pointeur, l'opération + replace X dans L (souvenons-nous que nous tenons à préserver cette valeur) et fait descendre automatiquement la pile, ce qui a pour effet de dédoubler non moins automatiquement la nouvelle adresse dans X.

...Et on recommence jusqu'à ce que toutes les valeurs de notre tableau aient été testées. Les pas 14 et 15 nous évoquent irrésistiblement le geste élégant du prestidigitateur qui, d'un mouvement sec et imperceptible, vous change une dame de pique en as de cœur. Finalement les instructions 18 à 21 (**CLX**, **LASTx**, **R↑**) remettent un peu d'ordre dans la pile, la valeur dans L retrouve sa place

en Y ; l'adresse du maximum est amenée en X et le maximum du tableau est placé en Z. Ces pas pourraient paraître superflus, il n'en est rien, leur rôle capital est de veiller sur votre confort, restaurer au maximum l'état antérieur de la pile et placer le résultat attendu là où on a coutume de retrouver tous les résultats : en X ! Cela, croyez-nous, vous évitera bien des migraines.

S'INTRODUIRE

Le lecteur attentif peut se demander d'où proviennent tous ces nombres que nous ne cessons de manipuler dans les entrailles de la mémoire ?

La réponse est fort simple : nous les y avons placés, et pour cela bien sûr, nous avons utilisé une procédure ! Pour introduire un tableau en mémoire on pourrait utiliser comme précédemment un contrôleur de boucle de type ddd, fff. Mais cela nous obligerait à connaître initialement la dimension, c'est-à-dire le nombre d'éléments, de notre vecteur. Ce paramètre peut très bien ne pas être immédiatement disponible et, quand il l'est, il peut y avoir une erreur dans le compte des éléments. Il n'y a rien de plus frustrant que de voir le calculateur attendre une $n + 1^{ième}$ donnée dont on ne dispose pas où, à l'inverse, nous abandonner lâchement bien qu'il nous reste encore des nombres à introduire.

La solution que nous adopterons sera de convenir d'un signe indiquant la fin de l'introduction. Dans ces conditions, le seul paramètre dont le calculateur aura besoin est l'adresse de la base du tableau.

Quel peut être ce signe ? Un chiffre clef comme zéro ?

Un nombre caractéristique comme π ? Si vous adoptez cette solution ayez la certitude, en vertu de la loi de la tartine de beurre (dite de l'emm... maximum !), que vous aurez un jour à introduire ces valeurs et qu'alors le programme s'achèvera sans se soucier de votre désespoir (notons que la chose survient en général lors d'une démonstration visant à prouver combien est fiable le programme).

Le deuxième écueil est de choisir une clef de terminaison trop compliquée, de l'oublier, et d'être forcé de vous replonger dans la liste du programme pour

retrouver (ah oui ! C'est vrai.) les dix derniers chiffres de votre numéro de sécurité sociale !

Une nouvelle fois restons logiques et simples : puisqu'il n'y a plus rien à rentrer, ne rentrons rien. Votre calculateur est suffisamment sophistiqué pour faire la différence entre ce rien et tout le reste. Il utilise pour cela un drapeau.

De même que le calculateur gère automatiquement de nombreux mouvements de la pile opérationnelle, il manipule aussi pour son propre compte certains indicateurs, parmi lesquels celui qui nous occupe : le drapeau 22. Ce dernier est toujours désarmé à la mise sous tension du calculateur. Dès qu'une entrée numérique est effectuée il s'arme sans que l'utilisateur ait à intervenir. Il suffit alors de tester l'état de l'indicateur pour que le programme puisse faire la distinction entre deux situations : une valeur vient d'être introduite ou non.

Après cette digression sur les indicateurs binaires, revenons à notre procédure d'entrée : Elle ne compte pas moins de 35 instructions, décidément nos programmes prennent de l'embonpoint.

Sa logique n'est pas cependant plus complexe que celles des procédures précédentes en voici la liste :

```
01*LBL"TEN"
02 XEQ"FM?"
03 STO T
04 RDN
05 FIX 0
06 CF 22
07 INT
08 SIGN
09 CLX
10*LBL 04
11 CLA
12 ARCL X
12 "⊢?"
14 PROMPT
15 FC?C 22
16 GTO 00
17 STO IND L
18 CLX
19 1
20 ST+Y
21 ST+L
22 RDN
23 GTO 04
24*LBL 00
25 FIX IND Z
26 DSE L
27 LASTX
28 -
29 CHS
30 LASTX
31 1 E3
32 /
33 +
34 ISG X
35 .END.
```

Sur le plan de l'architecture ce programme comporte une boucle du type TANT QUE . . . FAIRE. Le test de sortie au pas 15 précède en effet l'incrémentation de la boucle. Il est donc parfaitement possible de n'introduire aucune

valeur. Mais, d'une part, ce n'est pas le but recherché, et d'autre part le code de contrôle renvoyé est bien curieux, l'adresse du sommet est inférieur à celle de la base !



Et voici le mode d'utilisation : Avant l'appel de la procédure l'adresse de la base du tableau doit se trouver en X.

À chaque tour de boucle le calculateur s'arrête en affichant le rang de la donnée à introduire. S'il y a encore des entrées à effectuer, introduisez la valeur et frappez **R/S**, sinon il suffit de frapper directement **R/S**. La procédure s'achève avec en X le code de contrôle du tableau. Le registre Y est conservé.

En sus des instructions de commande et test des drapeaux, ce programme comporte des instructions de manipulation de chaînes de caractères. Le lecteur peu familiarisé avec ces dernières pourra se reporter au chapitre suivant.

Il est clair que la procédure que nous venons de développer pourrait être plus brève, mais ce serait au détriment de sa souplesse d'utilisation. C'est sur ce critère qu'elle a été optimisée : la conservation du format d'affichage en utilisant **FM?** mérite bien la perte de quelques octets.

L'optimisation du nombre de pas, primordiale aux temps héroïques des calculatrices de faible capacité, reste une bonne habitude. Elle doit cependant céder la place au confort. Si vous pouvez réduire le nombre de pas ou la taille d'une instruction sans modifier le mode d'utilisation ou augmenter le nombre de registres nécessaires, n'hésitez pas à le faire. Dans le cas contraire, ne prenez cette peine qu'en cas de force majeure : lorsque s'affichent à l'écran les fatidiques **"PACKING TRY AGAIN"** ce qui, compte-tenu de la capacité maximum de 2,2 Ko du HP-41 C/V (soit une moyenne de 1100 instructions programmables) devient rarissime.

À CORPS ET A TRI

Nous allons maintenant travailler nos tableaux au corps. Jusqu'à présent nous n'avons fait que les explorer, ou les créer. Nous allons maintenant les transformer.

Une opération très souvent effectuée sur un tableau est le tri. Une des plus nobles activités de l'humanité est d'ordonner, de diminuer l'entropie qui l'environne. Classer est à l'origine de toute science. En marge de ces considérations philosophiques il faut noter que trier des valeurs peut s'avérer extrêmement utile.

Les informaticiens depuis quelques 21 millions de minutes que cette noble corporation s'active, ont utilisé une part non négligeable de ce temps à mettre au point des algorithmes de tri : par insertion, par bulles ou par sélection, méthodes de SHELL et de SHELL-METZNER. Nous avons le choix des armes, aussi nous intéresserons nous surtout aux contraintes à imposer à notre procédure.

Elle doit, avant tout, respecter les standards que nous nous sommes efforcés d'établir : le passage en entrée du code de contrôle ddd, fff doit suffire pour désigner la zone de la mémoire sur laquelle va porter le tri. Comme corolaire nous nous interdirons d'utiliser un seul registre de donnée (en dehors bien entendu du bloc à trier), le mot d'ordre est encore et toujours *: "Tout dans la pile"*!

Nous vous proposons d'utiliser l'algorithme de sélection. C'est un compromis satisfaisant entre l'efficacité et la complexité. Il consiste à sélectionner le plus grand élément du vecteur et à échanger cet élément avec celui situé à la base du tableau. On considère alors le sous-tableau ayant un élément de moins que le précédent (l'élément déjà classé) et on répète l'opération. Les itérations s'achèvent quand le sous-tableau ne contient plus qu'un élément, obligatoirement le minimum.

Il ne vous reste plus qu'à programmer tout cela, essayez, c'est encore plus simple quand on sait que c'est possible.

Lorsque vous y serez parvenu, vous n'aurez plus qu'à sauter le paragraphe qui suit. À moins que la curiosité ne vous pousse à comparer votre œuvre à

celle que nous vous proposons (un détail pour stimuler cette curiosité notre procédure n'utilise que onze instructions !)

```
01*LBL"TRI"
02*LBL 05
03 ENTER↑
04 XEQ"TMX"
05 RCL IND X
06 X<> IND Z
07 STO IND Y
08 RCL Z
09 ISG X
10 GTO 05
11 .END.
```

Si nous avons écrit la procédure TMX c'est pour nous en servir ! Nous commençons à récolter les fruits de notre dur labeur. Connaissant l'algorithme, l'écriture de ce programme n'a pris que cinq minutes, ceci est possible car TMX est un sous-programme "propre" et qu'il préserve le contenu de Y.

Sur cet exemple on commence à apprécier la puissance du calculateur et de son langage : un investissement initial limité pour créer des procédures bien léchées est toujours rentable. Nous avons souvent entendu des programmeurs vanter les perfectionnements de leur langage... pour regretter aussitôt de ne pouvoir disposer de telle ou telle fonction et d'être contraint de réécrire dans chaque programme des sous-programmes parfois très complexes.

Avec HP-41 vous n'aurez jamais de tels regrets. Grâce à son architecture unique tous vos programmes pourront utiliser les procédures déjà écrites comme n'importe quelle fonction du langage.

À VOS RANGS...

Revenons à nos procédures. TRI classe les valeurs, mais leurs positions d'origine sont perdues, or pour certaines applications il est essentiel de conserver cette information. Si vous devez classer n joueurs en fonction de leur score la procédure TRI ne vous sera pas d'une grande utilité. Vous pourrez certes connaître rapidement le score du troisième joueur, mais vous ne pourrez plus dire qui est le troisième.

La procédure suivante : TRN (Tableau RaNg) va remplacer les valeurs

d'un tableau par leur rang sans modifier leur position. Cette procédure fonctionne avec satisfaction si les valeurs contenues dans le tableau sont toutes positives.

À chaque tour de boucle le calculateur sélectionne la plus grande valeur du tableau et la remplace par son rang affecté d'un signe négatif. C'est impératif si on veut que le calculateur ne mélange pas allègrement les valeurs non encore classées (positives) avec le rang de celles qui le sont déjà.

Cette procédure, vous vous en doutiez, fait à nouveau appel à **TMX**. Mais un léger problème se pose. **TMX** ne conserve qu'un seul registre et une brève analyse nous montre que nous aurons besoin de conserver deux valeurs : le code du tableau qu'utilise **TMX** et un compteur qui nous indiquera le rang de la valeur retenue. Allons-nous être contraints d'utiliser un registre mémoire pour stocker une de ces valeurs ? Il n'en est rien, avec un peu d'astuce il est possible de conserver toute l'information au prix d'une légère perte de temps.

Reprenons **TMX**. À la fin de cette procédure la pile contient les valeurs suivantes :

- T = Code de contrôle “consommé” ;
- Z = Valeur maximum ;
- Y = Registre sauvegardé ;
- X = dresse de la valeur maximum.

Le code dans T contient, dans sa partie décimale, mais aussi dans sa partie entière (à une unité près) l'adresse de fin du tableau. Si nous disposons par ailleurs du nombre d'éléments de ce dernier il nous est alors possible de reconstituer le code initial. Ce nombre d'éléments constituera la partie fractionnaire du compteur de boucle préservé en Y.

Le programme **RNG** s'il ne pose pas de problème "stratégique" est donc relativement complexe sur le plan tactique :

```
01*LBL"TRN"
02 ENTER↑
03 XEQ"TNB"
04 1 E3
05 /
06 1
07 +
08 X<>Y
09*LBL 06
10 XEQ"TMX"
11 RDN
12 INT
13 CHS
14 STO IND T
15 LASTX
16 RDN
17 +
18 CLX
19 RCL Z
20 FRC
21 1 E3
22 *
23 -
24 ISG Y
25 GTO 06
26 ABS
27 -1
28*LBL 07
29 ST*IND Y
30 ISG Y
31 GTO 07
32 LASTX
33 .END.
```

UN PROBLÈME DE TAILLE

Ne cherchez pas **TNB** parmi les procédures déjà écrites, vous ne la trouveriez pas. C'est maintenant à vous de la réaliser. Son but est de calculer le nombre d'éléments d'un tableau à partir de son code de contrôle.

Ses spécifications sont les suivantes :

- le code d'un tableau est initialement en X ;
- après exécution de **TNB** le nombre d'éléments du tableau a remplacé ce code ;
- les registres Y et Z sont préservés.

Ne regardez pas la suite avant d'avoir un peu souffert :

```
01*LBL"TNB"
02 INT
03 ST-L
04 1 E3
05 ST*L
06  X<> L
07 X<>Y
08 -
09 1
10 +
11 .END.
```

C'est une procédure très utile qui mérite d'être individualisée.

Un autre exercice intéressant consiste maintenant à suivre l'exécution de **TRN** et de comprendre comment il fonctionne. Pour être franc nous ne sommes pas très sûrs d'avoir nous-mêmes compris, et pourtant il tourne !

PAS DE BONHEUR SANS MÉLANGE

Après avoir mis de l'ordre dans nos tableaux, il est tentant de procéder à l'opération inverse qui consiste à les mettre en désordre. L'utilité de cette opération ne peut vous échapper si vous savez que le poker se joue avec cinq cartes ou si vous êtes un statisticien en quête de permutations aléatoires pour la réalisation d'un schéma d'expérience.

La structure du programme **TML** (TABLEAU MÉLANGE) est la copie conforme de la procédure **TRI**. Mais au lieu d'appeler **TMX** nous appellerons **GAG** : au lieu de sélectionner sur un critère précis l'élément à permuter, nous le choisissons au hasard.

```
10✱LBL"TML"
02✱LBL 08
03 ENTER↑
04 XEQ"GAG"
05 RCL IND X
06 X<> IND Z
07 STO IND Y
08 RCL Z
09 ISG X
10 GTO 08
11 .END.
```

Cet exemple nous montre la force des structures logiques et l'économie de temps et de moyens qu'apporte leur utilisation rationnelle. Tout ceci n'est possible que parce que les standards adoptés pour `HP-41` sont tout à la fois cohérents et harmonieux et que nos efforts portent sur le respect de ces normes.

Une programmation correcte dans un langage déterminé ne peut se faire qu'en suivant la pente naturelle du langage. Un algorithme limpide en langage APL pour `HP-41` peut aboutir à un programme incompréhensible s'il est traduit littéralement en BASIC ou en PASCAL.

S'AFFICHER

Après avoir inconsidérément modifié un tableau il est bon de pouvoir le visualiser, ne serait-ce que pour profiter des cogitations du calculateur.

La structure de **TAF** (TABLEAU AFFICHAGE) nous sera d'emblée familière : c'est un balayage simple du tableau analogue à ce que nous avons déjà vu pour **TCL** et **TSM**. Les seuls problèmes que nous rencontrerons sont liés à la présentation de l'affichage. Nous allons retrouver les instructions de manipulation de caractères et de drapeaux que nous avions déjà utilisées pour la procédure **TEN**.

Nous désirons obtenir le format d'affichage suivant :

```
NNN= Valeur
```

nnn sera le rang dans le tableau de la valeur actuellement affichée, il doit être chargé dans le registre alpha en **FIX 9**. La valeur correspondante sera par contre affichée avec le nombre de décimales défini avant l'appel de la procédure, pour cela, vous vous en doutiez, nous allons réutiliser **FM ?**

```
01*LBL"TAF"
02 XEQ"FM?"
03 RDN
04 SIGN
05 CLX
06*LBL 09
07 CLA
08 FIX 0
09 ARCL X
10 "⊢= "
11 FIX IND T
12 ARCL IND L
13 AVIEW
14 ISG X
15 STO X
16 ISG L
17 GTO 09
18 ST-L
19 X<> L
20 .END.
```

Si vous trouvez que les valeurs défilent trop rapidement à votre gré, souvenez-vous que vous pouvez armer le drapeau 21 avant d'exécuter cette procédure, l'instruction **AVIEW** provoquera alors un arrêt à l'affichage de chaque valeur.



COPIES CONFORMES...

Avant d'abandonner les tableaux à une dimension voici deux ultimes procédures qui opèrent sur deux tableaux. Le but de la première est de recopier un tableau en mémoire à partir d'une nouvelle adresse. Le code de contrôle du tableau initial est présent en Y et la base de sa copie en X. Une analyse rapide (et peut-être incomplète !) du problème nous amène à écrire le programme suivant :

```
01*LBL"TCO1"
02 SIGN
03*LBL 00
04 RCL IND Y
05 STO IND L
06 CLX
07 1
08 ST+L
09 ST+Y
10 RDN
11 ISG Y
12 GTO 00
13 DSE L
14 LASTX
15 -
16 CHS
17 LASTX
18 1 E3
19 /
20 +
21 2
22 +
23 .END.
```

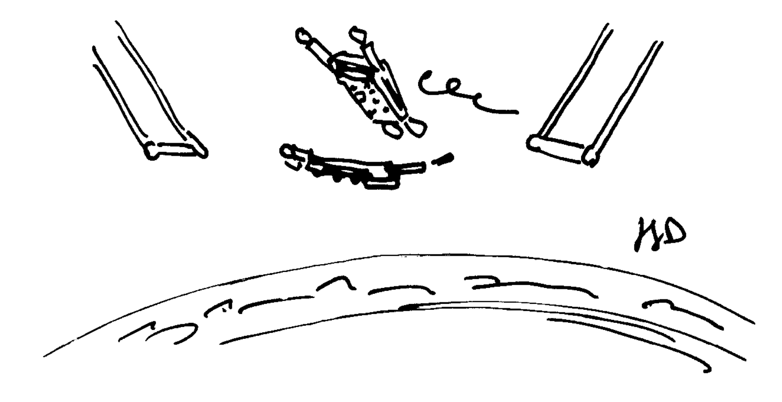
Cette procédure est satisfaisante si les aires occupées par les deux tableaux ne se chevauchent pas. Que se passe-t-il dans le cas contraire ?

Si l'adresse de base du tableau initial est inférieure à celle du tableau de départ ce dernier est en partie écrasé mais sa recopie est intégrale. Dans le cas contraire c'est la catastrophe ! La procédure modifie la partie supérieure du tableau, non encore recopiée, et on aboutit à une duplication d'une partie de l'information, le reste étant définitivement perdu.

Le programme **TCO1** est donc totalement proscrit dans ce cas de figure. Pour résoudre le problème dans cette situation il faudrait balayer le tableau source non plus de bas en haut mais de haut en bas.

Cette nécessité va nous conduire à étudier en détail une deuxième fonction de contrôle de boucle : **DSE** (Décrémentation et Saut si Egalité).

Comme **ISG**, la fonction **DSE** utilise un paramètre *ddd*, *fff*, ddd représentant cette fois l'adresse du haut du tableau. Mais le test de sortie étant un test d'égalité si *fff* est égal à l'adresse de la base du tableau ce dernier registre ne sera pas traité dans la boucle. Pour obtenir un balayage complet du vecteur le code de contrôle doit donc être égal à *ddd*,*fff*-1. Cette structure qui rompt un peu avec les normes que nous avons jusqu'ici rencontrées a probablement été introduite pour être compatible avec l'instruction **DSZ** (Décrémentation et Saut si Zéro) qui existait sur les calculatrices antérieures HP-67 et 97.



```
01*LBL"TCO"
02 X<=Y?
03 GTO 00                Test pour choisir le
04 1 E3                  sens de la recopie
05 X<>Y
06 *
07 LASTX
08 +
09*LBL 10
10 RCL IND Y             Transfert de bas en haut.
11 STO IND Y
12 CLX
13 1
14 +
15 ISG Y
16 GTO 10
17*LBL 11
18 INT
19 ST-L                  sous-programme d'invesion
20 1 E3                  du code de contrôle :
21 ST/Y                  ddd,fff→fff,(ddd-1)
22 ST*L
23 X<> L
24 +
25 RTN
26*LBL 00
27 X<>Y
28 XEQ 11
29 ENTER↑
30 XEQ"TNB"              Transfert de haut en bas.
31 ST-Z
32 RDN
33 X<>Y
34 1 E-3
35 X<>Y
36 *
37 LASTX
38 +
39*LBL 12
40 RCL IND Y
41 STO IND Y
42 CLX
43 1
44 -
45 DSE Y
46 GTO 12
47 1
48 +
49 .END.
```

Cette procédure, en plus de son intérêt utilitaire, peut constituer le cœur d'un embryon de base de données. La possibilité de déplacer une partie du contenu de la mémoire permet d'écraser un certain nombre de registres (décalage vers le bas) ou inversement d'ouvrir une nouvelle zone pour y insérer des données (décalage vers le haut) .

...ET LIBRE ÉCHANGE

Nous achèverons ce panorama des procédures de gestion de tableaux par un sous-programme aussi bref qu'utile, dédié aux nostalgiques de la fonction **P↔S** du HP-67.

Cette fonction permet d'échanger le contenu de deux tableaux indépendants. Les paramètres d'entrée sont, bien entendu, constitués par les codes de contrôle de ces tableaux, la procédure s'achève quand l'un de ces codes est "dépassé" :

```
01*LBL"T<>"
02 SIGN
03*LBL 13
04 CLX
05 RCL IND L
06 X<> IND Y
07 STO IND L
08 ISG Y
09 FS? 30
10 RTN
11 ISG L
12 GTO 13
13 .END.
```

DOUBLE ENTRÉE EN TRAITS SIMPLES

Nous venons de terminer l'étude des vecteurs à une dimension, et nous sommes en mesure de commencer à nous préoccuper de véritables tableaux à double entrée. De même que nous avons défini un tableau comme un ensemble de valeurs, on peut définir une matrice comme un ensemble de vecteurs, chaque vecteur constituant une ligne de la matrice.

Le problème est plus ardu que précédemment car une seconde dimension s'introduit dans l'espace mémoire linéaire du calculateur.

Heureusement nous ne sommes pas allés au bout des possibilités de l'instruction **ISG**, cette dernière peut en effet utiliser un incrément variable différent de 1. Pour illustrer cette capacité nous vous proposons l'expérience suivante :

En utilisant **TEN** chargez en mémoire à partir du registre R01 les chiffres de l à 8. La procédure renvoie le code du vecteur : 1,008. Ajoutez maintenant à ce code 2 E-5 et exécutez la procédure **TCL**. Si vous vérifiez maintenant le contenu des registres par la séquence : **1,008 XEQ "TAF"** vous pourrez observer que les nombres impairs et seulement eux ont été annulés ! La procédure **TCL** n'efface plus qu'un registre sur deux. La valeur $2E-5$ que nous avons ajoutée à la suite du code de contrôle est appe1lée le pas ou l'incrément. Nous pouvons représenter le vecteur initial sous la forme suivante :

R01	=	1	R02	=	2
R03	=	3	R04	=	4
R05	=	5	R06	=	6
R07	=	7	R08	=	8

ce qui peut être considéré comme une matrice à 4 lignes et 2 colonnes. Exécutée avec le paramètre $1,00802$ la procédure **TCL** n'a effacé que la première colonne.

Le code de contrôle complété par l'incrément recèle donc tous les éléments nécessaires et suffisants à la définition d'une matrice : son domaine d'implantation en mémoire caractérisé par l'adresse de base (ddd) et celle du sommet (fff) ainsi que son nombre de colonnes (cc). Comme nous l'avons montré dans l'exemple précédent, ce code de contrôle est en fait celui du premier vecteur colonne.

Ces prémisses achevées notre ligne de conduite sera simple : pouvoir utiliser toutes les procédures décrites pour les vecteurs sur les éléments (ligne, colonne ou totalité) d'une matrice.

Dans l'exemple ci-dessus, nous avons utilisé la procédure TCL sur la première colonne. De façon analogue le traitement d'un autre élément se fera en calculant préalablement, par une fonction, le code de contrôle spécifique de cet élément.

Certaines de ces fonctions génératrices de code de contrôle sont parfaitement triviales, ne comportant que de rares instructions. Elles pourraient être introduites en séquence dans le programme principal mais il est préférable pour des raisons de clarté de les conserver sous forme de procédures.

Contrôle de matrice : CMM

Si l'on souhaite balayer l'ensemble des éléments d'une matrice le plus simple est de la considérer comme un vecteur unique dont le code de contrôle est celui qui définit la matrice mais débarrassé de son incrément. Donc :

```
01*LBL"CMM"
02 1 E3
03 *
04 INT
05 1 E3
06 /
07 .END.
```

Sans commentaire !

Contrôle de colonne : CMC

Cette procédure (est-ce possible) est encore plus simple que la précédente : il suffit d'ajouter au code de contrôle matrice (en X) l'indice de 0 à $c-1$ de la colonne choisie (en Y) pour obtenir le code de contrôle de cette colonne :

```
01*LBL"CMC"
02 X<>Y
03 INT
04 +
05 .END.
```

Contrôle ligne : CML

Cette procédure est plus complexe que les précédentes car la totalité du code matrice initial est remanié :

```
01*LBL"CML"
02 X<>Y
03 INT
04 RCL Y
05 FRC
06 ISG X
07 INT
08 *
09 ST+Y
10 X<> L
11 X<>Y
12 INT
13 ST+Y
14 X<>Y
15 1 E-3
16 ST*Y
17 -
18 +
19 .END.
```

Nous attirons plus particulièrement votre attention sur les instructions 05 à 07. Elles permettent, sans modifier la pile (le registre L est cependant détruit), d'extraire rapidement le nombre de colonnes (l'incrément) du code de contrôle de matrice. L'avantage de ce processus est d'être valide même si l'incrément est absent, c'est-à-dire égal à 1 par défaut.

Un autre problème à résoudre au cours des manipulations de matrice est la localisation d'un élément à partir de ses coordonnées. La pratique a montré que la séquence suivante était raisonnablement efficace si le numéro de ligne est en Z, le numéro de colonne en Y et le code matrice en X.

```
01*LBL"CMP1"
02 XEQ"CML"
03 GTO"CMC"
```

Cette"procédure" renvoie non seulement en partie entière l'adresse de l'élément recherché mais restitue en fait le code de contrôle du segment de ligne qui va de l'élément au bord droit de la matrice, ce qui pourrait être utile

pour en explorer la partie triangulaire supérieure. Cependant, le contenu du registre T est perdu. Si vous tenez à le conserver, nous vous proposons une seconde version presque aussi brève :

```
01*LBL"CMP"
02 XEQ"CMC"
03 XEQ"CML"
04 INT
05 .END.
```

Pour achever ce groupe de procédures voici l'opération inverse de la précédente : à partir de l'adresse en Y et du code matrice en X retrouver les coordonnées du point.

```
01*LBL"COO"
02 INT
03 ST- Y
04 X<> L
05 FRC
06 ISG X
07 INT
08 XEQ"DIV"
09 X<>Y
10 .END.
```

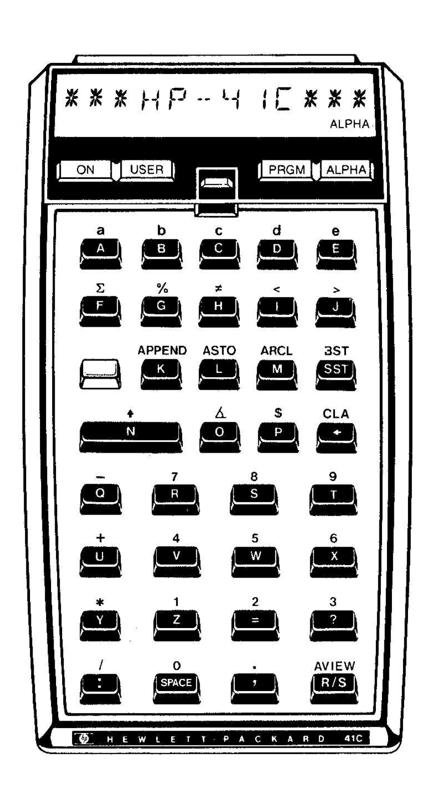
Grâce aux puissantes propriétés de DIV la procédure COO préserve les registres Z et T.

CONCLUSION

Nous avons essayé d'obtenir des procédures compactes et rapides. Avec les cinq dernières fonctions vous pouvez traiter les matrices avec une grande facilité, quel que soit leur emplacement en mémoire.

Pour être utilisées avec efficacité, ces fonctions exigent un certain entraînement. Mais après cette brève période d'adaptation vous pourrez aborder sans peur, tout problème de calcul matriciel, et créer des programmes sans reproche.

CHAPITRE VI
L'ALPHA ET L'OMÉGA





HP-41 est doté d'un écran "cristaux liquide" permettant l'affichage simultané de 12 caractères. Chaque caractère est formé sur une matrice de **14+3** segments. Toutes les possibilités d'affichage ($2^{14} = 16384$) n'ont pas été retenues. Seuls 60 caractères sont actuellement disponibles à l'affichage HP-41.

Vous avez sans doute noté qu'il était nécessaire de presser la touche **ALPHA** pour accéder à ces lettres et ces symboles.

Rappelons également qu'il suffit de penser à retourner le calculateur pour avoir l'image du clavier alpha !

L'utilisation des caractères alphanumériques se fait selon trois axes :

- la programmation ;
- des textes de commentaires ;
- la manipulation de chaînes de caractères.

LA PROGRAMMATION

À la base de toute écriture d'un programme sur HP-41 se situe l'obtention des mnémoniques correspondant aux fonctions. Il est donc indispensable d'avoir pratiqué les exemples du manuel de l'utilisateur et de s'être assuré au moyen de l'annexe 3 que toutes les méthodes d'obtention des fonctions ne posent plus de problèmes.

HP-41 peut faire des commentaires en cours de programme. C'est en effet un des rares micropoches capable d'afficher un texte tout en continuant à travailler. Cela évite de s'abîmer dans la contemplation de l'écran dans l'attente du flash d'information. De plus, cela peut constituer une aide importante à la mise au point des programmes. Il est utile d'indiquer l'opération que le calculateur effectue : JE TRIE, JE CLASSE, JE NUMEROTE, JE DORS...

D'autre part, n'oubliez pas de prévoir un temps d'affichage minimum (une seconde) pour permettre la lecture d'un message. Ce temps peut être allongé en intercalant des pauses (**PSE**) dans le cours du programme.

Enfin, il est bon d'insérer des remarques dans le corps du programme pour faciliter sa mise au point et sa maintenance. HP-41 ne dispose pas de fonction spécifique dans ce but, mais il est possible d'en obtenir un excellent succédané : il suffit de faire précéder la chaîne de caractères par l'instruction

FS ? 30. Ce drapeau étant, ainsi que nous l'avons déjà signalé, toujours, désarmé pour l'utilisateur, le message qui le suit ne sera jamais pris en compte lors de l'exécution du programme.

Si la liste des remarques est plus longue, par exemple lors de la description des variables en tête du programme, il est préférable de court-circuiter l'ensemble par un branchement inconditionnel (**GTO**).

```
01*LBL"EXEMPLE"
02 GTO 00
03 "R0=CODE MAT"
04 "R1=COMPTEUR"
05 "R2=VAL CHERCHEE"
06*LBL 00
07 .END.
```

LES TEXTES COMMENTAIRES

Ces commentaires constituent l'essentiel du dialogue entre vous et le calculateur.

HP-41 ***pose une question à laquelle l'utilisateur doit répondre pour que le programme se poursuive.***

La fonction à utiliser dans ce cas est **PROMPT**. Il est également possible d'utiliser la fonction **AVIEW** avec l'indicateur 21 préalablement armé, si vous envisagez d'utiliser un jour une imprimante et que vous souhaitez voir s'imprimer l'intitulé de la question.

Exemples :

– si la réponse est numérique :

```
01 "SEMENCE ?"
02 PROMPT
```

– si la réponse est un texte :

```
01 "OUI/NON ?"
02 AON
03 SF 21
04 AVIEW
```

Quelques conseils pour vos questions :

- Évitez les messages trop longs
  Exemple : **INENVISAGEABLE**
  qui après défilement n'a plus tout à fait le sens désiré.
- Évitez les messages trop courts
  Exemple : **?**
  qui est très Basic mais pas très explicite.
- Précisez, si possible, les unités ou les formats.
  Exemple :

```
"0<N<1 ?"
"TPS HH,MMSS ?"
"DATE JJ,MMAA"
```

- Pour les questions répétitives il est bon de créer une boucle qui précise le rang d'entrée de la valeur. C'est ce que nous avons fait avec la procédure **TEN** (78) et ce que nous ferons en atteignant **ATEN**.

Un dernier cas à envisager est celui des *menus*. Un menu est un ensemble d'options proposées à l'utilisateur qui doit en sélectionner une. L'affichage de HP-41 étant constitué d'une unique ligne, les options seront proposées séquentiellement :

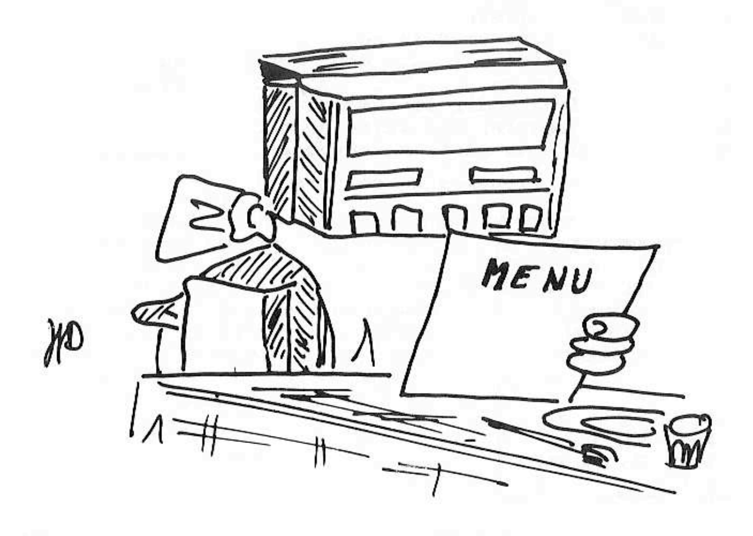
```
10 "1-ENREGIS"
11 PSE
12 "2-CALCULS"
13 PSE
14 "3-FIN"
15 PSE
16 "OPTION<1-3>?"
17 TONE 9
18 PROMPT
19 XEQ IND X
```

Cette présentation est cependant assez lente et peut finir par lasser l'utilisateur averti qui connaît par cœur les codes des options.

Il est préférable d'utiliser la seconde formule que voici :

```
08♦LBL 01
09 CF 22
10 "OPTION<1-3>?"
11 TONE 9
12 PROMPT
13 FS?C 22
14 GTO IND X
15 "1-MOUVEMT"
16 PSE
17 "2-RADAR"
18 PSE
19 "3-TIR"
20 PSE
21 GTO 01
```

Si l'utilisateur frappe **R/S** sans introduire de valeur alors le menu lui sera présenté puis la question à nouveau posée.



`HP-41` ***vous répond***

Les conseils, que nous vous donnerons, sont le reflet de ceux proposés pour les interrogations.

Les réponses seront documentées de la même façon, en précisant, si nécessaire, les unités choisies.

Une réponse est généralement constituée de trois éléments :

– le descriptif indiquant succinctement de quoi il s'agit ;
– une valeur numérique ;
– les unités.

Ces trois éléments sont placés successivement (voir plus loin la concaténation, page :103) dans le registre ALPHA par une séquence d'instructions du type :

```
01  "TEMP"
02  ARCL X
03  "⊢ DEG"
04  AVIEW
```

Il est important de se souvenir que la fonction **ARCL** place dans le registre ALPHA un nombre dans le format (**FIX**, **SCI**, **ENG**) et avec le nombre de décimales actuellement sélectionné. Il faut donc vérifier que ce format permet d'obtenir un affichage propre. Il est également souhaitable d'arrondir le nombre à afficher (**RND**) pour éviter de contempler une suite de décimales, résultat des approximations parfois douteuses du calculateur.

Voici d'autres exemples de sorties :

```
6.5 METRES
TAXE 17%
R2= 10 OHMS
SANS RACINE
SANS CORNEIL
PRX : 100 F


FACT Z : OK
XYZ EN ALM
VOL 3,14 M↑3
```

Remarque : Il est toujours utile de compléter l'affichage par un signal sonore qui attire l'attention et souligne le texte. Nous proposons un son grave pour les indications d'erreur et un son aigu pour les messages interrogatifs.

STRUCTURE DU REGISTRE ALPHA

Contrairement à la petite phrase du manuel de l'utilisateur p36, le registre ALPHA n'est pas équivalent au registre X de la pile opérationnelle. Pour

mieux saisir ces différences, nous allons effectuer quelques manipulations. Placez préalablement

"A" dans R01 : **ALPHA** **A ASTO 01**

"B" dans R02 : **B ASTO 02** **ALPHA**

Effectuez maintenant la séquence : **RCL 01 RCL 02 STO 02 VIEW 02**

Le registre 02 contient toujours "B". Nous allons effectuer une séquence apparemment semblable avec :

ALPHA **ARCL 01 ARCL 02 ASTO 02** **ALPHA** **VIEW 02**

Le registre 02 contient maintenant **"AB"**. Il est visible que les deux suites d'instructions ne sont pas équivalentes. D'une part pour le registre ALPHA il y a eu *concaténation* (voir plus loin le détail de ce travail, page : 103), d'autre part la pile opérationnelle a une structure

LIFO (Last In First Out) en français : « Les derniers seront les premiers ». Le registre ALPHA, quant à lui est du type FIFO (First In First Out) en français : « Faites la queue comme tout le monde ! »

Mais ce n'est pas la seule subtilité ; reprenons l'exemple du manuel p. 36 :

ALPHA **"Ex"** **SPACE** **ASTO 01 "DE"** **SPACE** **ASTO 02 "DECALAGE" ASTO 03** **CLA**

ARCL 01 ARCL 02 ARCL 03

On obtient à l'affichage : "**X DE DECALA** "

Il y a de la fuite dans les idées !

Que nous apprend cet exemple ? Chaque registre contient au plus 6 caractères (manuel page 55) c'est pourquoi nous perdons le "GE" de "DECALAGE".

Le "E" de "EX" est lui présent dans le registre ALPHA mais n'apparaît pas à l'affichage limité à 12 caractères. Une double-pression sur le **ALPHA** ramènera à l'écran en éliminant le tiret d'attente.

ON COUPE ET ON COLLE

Envisageons maintenant les chaînes de caractères comme des objets que HP-41 peut manipuler au même titre que les valeurs numériques. Les trois seules opérations possibles sur les chaînes sont la *troncature*, la *concaténation* et les *tests*.

La troncature

Nous venons d'observer le phénomène dans l'exemple qui précède. Il nous reste à dompter cet animal un peu rétif. En examinant les possibilités offertes pour tronçonner les chaînes il apparaît à l'évidence que dans sa version de base `HP-41` n'a pas été conçu pour le traitement de texte.

Il existe deux façons de découper le contenu du registre ALPHA : le stockage dans une mémoire et la fonction **ASHF** (ALPHA SHIFT : glissement du registre ALPHA). Dans les deux cas la césure se fait entre le sixième et le septième caractère. Travaillant ensemble ces deux fonctions permettent cependant de stocker en mémoire la totalité du registre ALPHA.

Pour illustrer la chose, et puisque nous ne vous avions point proposé de procédure depuis fort longtemps, en voici une qui vous permettra de placer en mémoire une succession de chaînes.

```
01*LBL"ATEN"
02 .1
03 %
04 +
05 XEQ"FM?"
06 FIX 0
07 X<> Z
08 1
09 -
10 6
11 /
12 INT
13 SIGN
14 ST+ L
15 CLX
16 STO T
17 CF 23
18*LBL 01
19 "CHN"
20 ARCL T
21 "⊢?"
22 AON
23 PROMPT
24 FC?C 23
25 GTO"CODE"
26 CLX
27 LASTX
28*LBL 02
29 ASTOIND Y
30 ASHF
31 ISG Y
32 STO X
33 DSE X
34 GTO 02
35 ISG T
36 STO X
37 GTO 01
38*LBL"CODE"
39 FIX IND Z
40 X<> L
41 1E2
42 /
43 X<>Y
44 INT
45 ST- L
46 ST+ Y
47 CLX
48 1E3
49 ST* L
50 ST/ Y
51 X<> L
52 +
53 AOFF
54 .END.
```

Cette procédure est longue mais elle fait beaucoup de choses :

- Les paramètres d'entrée sont le registre de base de la matrice de texte en X et le nombre maximum de caractères d'une chaîne. Ce dernier paramètre va servir à déterminer le nombre de registres nécessaires au stockage d'une ligne de texte.
- Les pas 0l à 18 sont destinés à diverses opérations, mettre en mémoire le format, préparer les pointeurs, déterminer le nombre n de registres affectés à une ligne, et passer en mode alpha.

- Du pas 19 au pas 38 se situe la boucle d'interrogation, sa structure est très proche de celle de **TEN**, ici encore la terminaison des entrées se fait en pressant directement **R/S** mais c'est cette fois le drapeau 23 qui est testé.
- Les instructions 29 à 35 constituent une seconde boucle, imbriquée dans la première, dont le rôle est de découper le contenu de ALPHA et de le charger dans N registres consécutifs.
- Les lignes 39 à 55 terminent la tâche en calculant le code de la matrice finale dont le nombre de lignes est égal au nombre de lignes de texte introduites et le nombre de colonnes (l'incrément du code de contrôle) précise le nombre de registres requis pour la mise en mémoire d'une ligne.

La concaténation

La nécessité de reconstituer les chaînes est évidente. Cette opération, la concaténation, consiste donc à enchaîner les chaînes les unes derrière les autres.

Si HP-41 tronque mal, il concatène fort bien grâce aux deux "instructions" **ARCL** et "⊢". Si nous avons mis des guillemets à *instructions* c'est pour préciser que n'est pas une instruction mais un caractère.

Un peu particulier ce caractère puisqu'il n'apparaîtra jamais à l'affichage. Placé au début d'une ligne de texte en mode programme il indique que les caractères qui le suivent doivent se placer à la suite du contenu courant de ALPHA et non le remplacer. En mode direct cette instruction place derrière la chaîne affichée le tiret d'attente, le contenu de ALPHA peut alors être complété ou corrigé.

La fonction **ARCL** cache elle aussi des propriétés particulières, c'est bien plus qu'une instruction de rappel du contenu d'un registre. Si le registre adressé contient une valeur numérique **ARCL** se livre à un intense travail de codage pour transformer l'expression numérique de ce nombre en son image en code ASCII.

Si vous effectuez la séquence d'opérations suivante :

XEQ "CLA" FIX 4 π ALPHA ARCL X ASTO X 1/x

le calculateur vous répond ALPHA, le 3,1416 qu'affiche le registre X n'est plus un nombre mais la suite de six caractères 3, la virgule, 1 etc...

Ces notions sont arides, faisons une pause pour s'amuser un peu. Voici un petit programme qui vous permettra de faire pénétrer HP-41 dans le monde de la littérature.

Le *cadavre exquis* est un "exercice de style" qui fit fureur à la période faste de l'OULIPO (OUvroir de LItérature POtentielle, nous ne sommes pas les seuls à user d'abréviations barbares) auquel participèrent des noms célèbres comme Georges Perec, Raymond Quenaux.

Tombé un peu en désuétude si ce n'est dans le domaine enfantin, il s'agit pour un groupe de personnes d'écrire chacun un morceau de phrase dans l'ignorance de ce qui précède et de ce qui va suivre.

La phrase achevée, on place le tout bout à bout et on lit à haute et intelligible voix le résultat final. Ce dernier a, généralement, la clarté d'un vers de Nostradamus ou d'un message de Pierre Dac émis par la BBC durant la dernière guerre.

HP-41 grâce à sa prodigieuse faculté d'oublier ce qu'il vient d'exécuter quelques millisecondes plus tôt, pourra tout seul créer pour vous ces phrases inoubliables.

```
01*LBL"KDVX"          16 X<Y?
02 XEQ"HSD"           17 "⊢LE "
03*LBL 14             18 X>Y?
04 ADV                19 "⊢LA "
05 CLA                20 ARCLIND Y
06 XEQ"NOM"           21 RTN
07 XEQ"VERB"          22*LBL"VERB"
08 XEQ"NOM"           23 1.087
09 AVIEW              24 XEQ"GAG"
10 GTO 14             25 "⊢ "
11*LBL"NOM"           26 ARCLIND X
12 1.087              27 "⊢ "
13 XEQ"GAG"           28 RTN
14 ENTER↑             29 .END.
15 62
```

```
VOICI QUELQUES EXEMPLES        R18=  "COLLE"
                               R19=  "CORNE"
LE ZEBRE SOLDE LA PINCE        R20=  "COTE"
LA BASE BOSSE LA GRELE         R21=  "COUPE"
LA CACHE JURE LE COCHE         R22=  "DALLE"
LE SOLDE COCHE LA FILE         R23=  "DAME"
LA GARE PERLE LA PLUME         R24=  "DATE"
                               R25=  "FERME"
                               R26=  "FICHE"
R01=  "BANDE"                  R27=  "FILE"
R02=  "BARRE"                  R28=  "FORCE"
R03=  "BASE"                   R29=  "FORME"
R04=  "BECHE"                  R30=  "FOULE"
R05=  "BINE"                   R31=  "GARE"
R06=  "BISE"                   R32=  "GLACE"
R07=  "BORNE"                  R33=  "GOMME"
R08=  "BOSSE"                  R34=  "GRELE"
R09=  "BOTTE"                  R35=  "GREVE"
R10=  "BRIDE"                  R36=  "HACHE"
R11=  "CACHE"                  R37=  "JAUGE"
R12=  "CALE"                   R38=  "JOUE"
R13=  "CORDE"                  R39=  "LANCE"
R14=  "CASE"                   R40=  "LAVE"
R15=  "CASSE"                  R41=  "LIME"
R16=  "CAUSE"                  R42=  "MINE"
R17=  "CIRE"                   R43=  "NICHE"
```

…/…

```
(suite)
R44= "NORME"
R45= "NOTE"
R46= "PECHE"
R47= "PEINE"
R48= "PERLE"
R49= "PINCE"
R50= "PISTE"
R51= "PLUME"
R52= "POCHE"
R53= "PORTE"
R54= "RIDE"
R55= "RUINE"
R56= "RUSE"
R57= "SCIE"
R58= "SOUDE"
R59= "TACHE"
R60= "TAXE"
R61= "TRACE"
R62= "BARDE"
R63= "BRAVE"
R64= "COCHE"
R65= "CODE"
R66= "CORSE"
R67= "CREPE"
R68= "GARDE"
R69= "GIVRE"
R70= "GUIDE"
R71= "JOINT"
R72= "JUGE"
R73= "JURE"
R74= "LIVRE"
R75= "MIME"
R76= "REVE"
R77= "SABLE"
R78= "SABRE"
R79= "SACRE"
R80= "SIGNE"
R81= "SINGE"
R82= "SOLDE"
R83= "SUCRE"
R84= "VIDE"
R85= "VOILE"
R86= "ZEBRE"
```

Plus prosaïquement voici une nouvelle procédure qui vient compléter **ATEN** puisqu'elle permet, à partir du code de contrôle généré par cette dernière, de reconstituer et d'afficher les lignes précédemment mises en mémoire.

```
01*LBL"ATAF"
02 ENTER↑
03 XEQ"CMM"
04 X<>Y
05 FRC
06 ISG X
07 INT
08 SIGN
09*LBL 03
10 CLA
11 CLX
12 LASTX
13*LBL 04
14 ARCLIND Y
15 ISG Y
16 FS? 30
17 GTO 00
18 DSE X
19 GTO 04
20 AVIEW
21 TONE 7
22 GTO 03
23*LBL 00
24 AVIEW
25 TONE 7
26 .END.
```

Nous retrouvons des techniques éprouvées pour traitement des tableaux : l'appel à **CMM** et l'extraction de l'incrément par la séquence **FRC**, **ISG X**, **INT**. Nous attirons également votre attention sur l'utilisation du négateur **FS? 30** pour inverser le sens de sortie par **ISG** au pas 15.

Pouvons-nous aller plus loin dans le traitement des chaînes ? La réponse

est positive si on sacrifie deux facteurs importants : l'espace et le temps. Un tel sacrifice est, bien sûr, relatif.

Le but de la procédure qui suit est de décomposer à nouveau le registre ALPHA pour constituer un vecteur de caractères, un registre pour chaque caractère (c'est le sacrifice d'espace) et malgré les efforts désespérés du programmeur, cette procédure n'est pas très rapide (c'est le sacrifice de temps).

Cette procédure travaille entièrement dans la pile mais, nouvelle concession, elle ne peut traiter que des chaînes n'excédant pas 18 caractères :

```
01♦LBL"A-C"
02 INT
03 .1
04 %
05 +
06 SIGN
07 ASTO Y
08 ASHF
09 ASTO Z
10 ASHF
11 ASTO T
12 CLA
13 ASTO X
14 XEQ 05
15 XEQ 05
16 XEQ 05
17 LASTX
18 FRC
19 ST- L
20 1 E3
21 ST* Y
22 X<> L
23 X=Y?
24 ASTOIND X
25 X≠Y?
26 DSE X
27 LASTX
28 /
29 +
30 RTN
31♦LBL 05
32 X=Y?
33 GTO 00
34 "12345"
35 ARCL Y
36 ASTOIND L
37 ASHF
38 ASTO Y
39 "6"
40 ARCLIND L
41 ASHF
42 ASTOIND L
43 ISG L
44 STO X
45 GTO 05
46♦LBL 00
47 RDN
48 .END.
```

La procédure reçoit en X l'adresse de base du futur tableau de caractères, et dans ALPHA la chaîne à décomposer.

Les pas 02 à 13 constituent une séquence d'initialisation : préparation et stockage dans L de ce qui deviendra le code de contrôle, chargement dans les registres Y, Z et T du contenu de ALPHA (d'où la limite à 18 caractères) et création d'une chaîne vide dans X (il ne faut pas confondre une chaîne vide de zéro caractère avec un espace qui lui, bien que tout aussi invisible, est un caractère.

Comme il n'y a plus de place dans la pile pour y installer un compteur, un sous-programme 00 est appelé trois fois pour traiter le contenu de chacun des registres Y, Z, T. Après ces trois appels le code de contrôle final est généré. Cette séquence tient compte également du cas particulier du registre ALPHA initialement vide, dans ce cas, le tableau est réduit à un seul registre qui contient un caractère vide. C'est une option. On aurait pu choisir de signaler ce cas en renvoyant **0** en X à la place d'un code de contrôle. Vous choisirez en fonction de votre application.

Venons-en maintenant au sous-programme 00 qui constitue le cœur de l'ensemble, il est lui-même constitué d'une boucle qui traite les uns après les autres les caractères présents dans le registre Y.

Cette boucle est du type `TANT QUE (CONDITION) FAIRE`, le' test de sortie (**X=Y?** : le registre Y est-il vide ?) est donc placé en début de boucle.

Puis des pas 34 à 42 prend place une série de concaténations et de troncatures dont le but est d'isoler le caractère le plus à gauche du registre Y et de replacer dans ce même registre la chaîne initiale débarrassée de ce caractère. En regard des instructions du programme nous avons figuré l'évolution du contenu du registre ALPHA. On constate que le registre mémoire adressé par L, qui doit finalement recevoir le caractère isolé, est utilisé transitoirement comme registre tampon.

Une fois converti en tableau de caractères, et disposant du code de contrôle de ce dernier, quelles opérations peut-on effectuer ?

- Connaître la longueur de la chaîne : il suffit de placer le code de contrôle en X et d'appeler **TNB** ;
- Replacer la totalité de la chaîne dans ALPHA. Il nous faudra pour cela créer une procédure très simple :

```
01*LBL"C-A"
02 STO L
03 CLA
04*LBL 06
05 ARCLIND L
06 ISG L
07 GTO 06
08 .END.
```

- Le code de contrôle est initialement en X. En fin de procédure la chaîne

est reconstituée dans ALPHA, seul le registre L a été modifié.

– Il est également possible de ne rappeler dans ALPHA qu'une fraction de la chaîne initiale, c'est ce que fait la procédure suivante :

```
01*LBL"AFRC"
02 INT
03 SIGN
04 ST+ L
05 CLX
06 1 E-3
07 X<> L
08 ST* L
09 X<> L
10 +
11 GTO"C-A"
12 .END.
```

– Le registre X contient initialement le code de contrôle du vecteur de caractère, le registre Y contient lui un code $dd,0ff$ indiquant les rangs du premier et du dernier caractère de la sous-chaîne. En fin d'exécution cette sous-chaîne est dans ALPHA et dans X on retrouve son code de contrôle.

Voici un bref exemple d'utilisation de ces procédures :

ALPHA DECAMETRE ALPHA 2 XEQ "A-C"

HP-41 vous répond 2,010 vous indiquant que les caractères de la chaîne sont conservés dans les registres 02 à 10. Le registre ALPHA contient maintenant “E”, dernier caractère du mot introduit. Sauvegardons le code de contrôle dans R01 (**STO 01**) et appelons **TNB**. La réponse quasi immédiate est 9, effectivement “DECAMETRE” comprend bien neuf lettres !

Introduisons maintenant la séquence : **6,009 RCL 0 XEQ "AFRC"** . Un coup d'œil dans le registre ALPHA, on y lit “ETRE”. Poursuivons par **4,006 RCL 01 XEQ "AFRC"**, nous obtenons cette fois “AME”. Vous doutiez-vous qu'un banal instrument de mesure contenait potentiellement de tels concepts philosophiques ?

TESTER LES CHAÎNES

Les tests de chaînes, comme les numériques s'effectuent à l'aide des registres X et Y. On en déduit donc que, ne pourront être testées, que des chaînes de longueur inférieure ou égale à six caractères.

Seuls sont disponibles les tests d'égalité ou d'inégalité (**X=Y?** et **X≠Y?**) le tri alphabétique pour être réalisable doit faire appel à des techniques sophistiquées qui dépassent largement le cadre de cet ouvrage (nous renvoyons le lecteur intéressé aux articles et ouvrages donnés en bibliographie).

Ceci étant, toutes les méthodes décrites dans le deuxième chapitre concernant les tests d'égalité sont entièrement transposables aux chaînes, en particulier, les techniques de branchement indirect sont valides puisque les étiquettes peuvent être alphabétiques aussi bien que numériques.

Voyons-en un exemple très utile avec la procédure suivante.

Le choix d'une option se ramène souvent à une simple alternative qui demande comme réponse : *OUI* ou *NON*. Quand le problème se pose il suffit de placer dans le registre ALPHA l'intitulé de la question et de le faire suivre par **XEQ "O/N"**, au retour de la procédure l'indicateu1 **23** est armé si l'utilisateur a répondu "O" et désarmé s'il a répondu "N" :

```
01*LBL"O/N"       09 PSE            16 GTO 01
02 AON            10 FC? 23         17*LBL"N"
03 CF 21          11 GTO 02         18 CF 23
04*LBL 01         12 ASTO L         19*LBL"O"
05 CF 23          13 SF 25          20 CF 25
06 "⊢<O/N>?"      14 GTO IND L      21 AOFF
07 AVIEW          15 CLA            22 .END.
08*LBL 02
```

Le drapeau 23 est utilisé pour vérifier qu'une entrée a été effectuée, dans le cas contraire, la boucle sur l'étiquette **LBL 00** se poursuit et le message clignote à l'affichage. Si un caractère est introduit le drapeau **23** s'arme, le caractère est placé dansL et l'appel indirect est effectué pour un branchement à "N" qui désarme l'indicateur **23**, ou à "O" qui le conserve. Le tout s'effectue sous la haute garde de l'indicateur **25** qui veille à la validité des entrées.

Pour être exhaustif voici la façon de résoudre un ultime problème : les registres du HP-41 sont banalisés, ils peuvent contenir aussi bien une valeur numérique qu'un fragment de chaîne. Si vous désirez savoir, en cours de programme, si un registre contient l'une ou l'autre de ces deux catégories de variable il vous suffit de rappeler en X et d'effectuer la fonction **SIGN**. Dans le cas d'une chaîne le contenu de X sera nul, la chaîne étant sauvegardée dans L.

La fonction **SIGN** est la seule qui soit susceptible d'agir sur les chaînes comme sur les nombres.

ANNEXES

Solution des exercices

Obtention des fonctions

Description des fonctions

Drapeaux

Procédures et code-barre

ANNEXE 1
Solution des exercices

SÉQUENCE DE LETTRES NON RÉPÉTITIVES

Quatre lettres sont proposées, donc deux drapeaux suffiront pour mémoriser la dernière lettre introduite en utilisant le code suivant :

Lettre	F01	F02
A	0	0
B	0	1
C	1	0
D	1	1

Les 4 sous-programmes sont alors :

```
01 LBL A
02 FC?C 01
03 FS? 02
04 "⊢A"
05 CF 02
06 AVIEW
07 RTN
08 LBL B
09 FC?C 01
10 FC? 02
11 "⊢B"
12 SF 02
13 AVIEW
14 RTN
15 LBL C
16 FC?C 02
17 FC? 01
18 "⊢C"
19 SF 01
20 AVIEW
21 RTN
22 LBL D
23 FS? 01
24 FC? 02
25 "⊢D"
26 SF 01
27 SF 02
28 AVIEW
29 END
```

La fonction booléenne utilisée est un **NAND** (**NON ET**), l'instruction "⊢" n'est effectuée que si **NON (F01 et F02)** ce qui peut aussi s'écrire **NON F01 OU NON F02** ou encore **F01 implique NON F02**... et c'est pourquoi votre HP-41 est muette.

LA SUITE MYSTERIEUSE

Le fait que ce problème soit posé comme application du changement de base en facilite grandement la résolution.

La suite est constituée par différentes représentations du nombre *12* dans les bases de *12* à *4*. La valeur à trouver est donc l'expression de *12* en base *5* :

5 ENTER↑ 12 XEQ "10-b" → *22*

Moralité : en cas de problème il est bon de consulter la base.

AU PIQUET !

La solution est, pour le probabiliste moyen, absolument évidente :

– le nombre de mains possibles est égal aux nombres de combinaisons de 12 éléments parmi 32, donc :

  32 ENTER↑ 12 XEQ "CB" → *225792840*

– si on retire les têtes, soit 12 cartes, il en reste 20. Le nombre de mains justifiant l'annonce "10 de blanc" est donc :

  20 ENTER↑ 12 XEQ "CB" → *125970*

Il suffit de faire maintenant la division de ces deux valeurs pour découvrir que l'annonce se produit en gros une fois sur 1800 mains distribuées. Pour deux joueurs, à raison d'une donne toute les 5 minutes, cet événement se produit en moyenne une fois toutes les 70 heures. Bonne nuit !

P.S. Ce problème est extrait du livre : "La chance et les jeux de hasard" de Marcel Boll (sic) !

LA SOLUTION AU NOMBRE DE SOLUTIONS

Ce problème est dû à Pierre Berloquin responsable, entre autres, des jeux mathématiques du Monde.

La solution jaillit quand on découvre que **13** peut en fait s'écrire :

1 + 1+ 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 = 13

Parmi ces 12 signes + il suffit d'en sélectionner 3 pour isoler 4 sommes de **1** (c'est-à-dire 4 entiers). Si nous choisissons le 4e, le 7e et le 8e, nous obtenons une solution :

4 + 3 + 2 + 4 = 13

Il y a donc autant de solutions que de combinaisons de **3** signes **+** parmi **12** donc :

12 **ENTER↑** **3 XEQ "CB"** → *220*

Pour les anxieux, la liste des **220** solutions a été déposée sous pli scellé chez Maître Dumatheux.

ANNEXE 2

Obtention des fonctions

LA FONCTION XEQ

La méthode universelle pour obtenir la totalité des fonctions consiste à utiliser **XEQ**.

Le processus est le suivant :

– en mode calcul, pressez la touche XEQ immédiatement suivie de ALPHA ;

– épelez maintenant le nom de la fonction désirée (uniquement le nom, tel qu'il apparaît au catalogue **3**) et quittez le mode ALPHA.

Ex : XEQ ALPHA **SIN** ALPHA ;

XEQ ALPHA **FC ?** ALPHA.

Si la fonction ne demande pas de paramètre (cas de **SIN** dans l'exemple qui précède, elle est immédiatement exécutée en mode calcul ou enregistrée en mode programmation.

Si la fonction est paramétrée (cas de **FC ?**) le calculateur affiche alors de un à trois tirets d'interrogation pour l'introduction de ce paramètre.

Dans ce cas plusieurs possibilités sont à envisager !

– il s'agit d'un appel indirect : pressez immédiatement la touche jaune. Le calculateur affiche **IND** à la suite du nom de la fonction toujours suivi des tirets interrogatifs ;

– il s'agit d'un adressage direct. Trois cas sont alors possibles :

  – le paramètre est numérique. Introduisez le nombre de chiffres exigé. Dès l'entrée du chiffre ultime la fonction est exécutée ou mémorisée. Si le paramètre est compris entre **01** et **10**, les **10** touches supérieures du clavier peuvent être utilisées ;

  – le paramètre est alphanumérique (nom de procédure ou de programme). Pressez la touche ALPHA, frappez le nom désiré, puis à nouveau ALPHA qui déclenche l'exécution ou la mise en mémoire ;

  – le paramètre est l'indication d'un registre de la pile. Pressez tout d'abord la touche ? (point décimal), le calculateur affiche **ST**(pour STACK : en français PILE) suivi d'un seul tiret. Il vous reste à préciser

le registre voulu en appuyant sur la touche correspondante :

- STO pour le registre L
- 6 pour le registre X
- X pour le registre Y
- 1 pour le registre Z
- 9 pour le registre T

L'avantage de la fonction **XEQ** est sa généralité. Son utilisation présente cependant quelques inconvénients :

– la frappe de l'intitulé complet de la fonction peut devenir fastidieuse si le mnémonique est long et si la fonction est fréquemment utilisée ;

– la recherche de la fonction appelée peut être relativement longue si plusieurs modules d'application sont connectés.

L'ASSIGNATION

Pour éviter ces deux inconvénients il est possible d'appeler le nom d'une fonction en ne pressant qu'une seule touche. Le nom de cette fonction doit avoir été préalablement assignée à l'aide de la commande **ASN** :

- une assignation commence par la séquence de touche :

[■] [ASN] [ALPHA]

à la suite de cette séquence le nom de la fonction, tel qu'il apparaît au catalogue 3, est épelé. Si la fonction demande un paramètre, ce dernier ne peut pas être précisé au moment de l'assignation.

Quand l'intitulé de la fonction a été intégralement introduit, on presse à nouveau la bascule [ALPHA].

HP-41 affiche alors un tiret interrogatif. Il ne vous reste plus qu'à presser la touche à laquelle vous souhaitez affecter la fonction, pour achever l'assignation. Le nombre qui apparaît alors à l'écran indique la localisation de la touche choisie : le chiffre des dizaines précisant sur le clavier la ligne considérée et celui des unités le rang sur cette ligne de la touche choisie. Si la touche SHIFT a été utilisée, le nombre est alors négatif.**68**

Le rappel d'une fonction assignée peut se faire à tout moment à condition que le calculateur soit en mode USER. Dans le cas où la fonction appelée exige un paramètre ce dernier sera introduit à la suite du nom de la fonction comme il est indiqué au paragraphe précédent.

LES OPTIONS DÉFAUT

Pour vous simplifier la tâche, Hewlett-Packard, a préassigné au clavier **68** fonctions et commandes directement accessibles. Presser la touche correspondante, éventuellement après la touche ■ (SHIFT) reste la plus simple méthode d'appel d'une fonction.

ANNEXE 3

Description des fonctions

Index des fonctions

Toutes les fonctions du HP-41C peuvent être enregistrées comme instructions dans la mémoire programme sauf celles marquées d'un astérisque. Les fonctions apparaissant différemment au clavier et à l'affichage sont listées avec leurs deux formes, celle du clavier en premier.(Ex.: [√x] au clavier et [SQRT] à l'affichage). Sauf indication contraire (°), toutes les fonctions peuvent être exécutées à l'affichage et réaffectées.

Utilisation, modes et indicateurs

* ° [ON]	Touche de mise sous tension. Voir [OFF] et [ON] plus loin	
[OFF]	Mise hors tension	
[BEEP]	Signal sonore	
[TONE] n	Signal à tonalité variable.n= choix de la tonalité	
* ° [USER]	Touche du mode personnel	
* ° [PRGM]	Touche du mode programme	
* ° [ALPHA]	Touche du mode alphanumérique	
[XEQ] nom	Exécution.Nom de programme, de fonction,label numérique ou adresse indirecte	**[XEQ] ([ALPHA] ajoute un octet pour chaque caractère du nom)** **[XEQ] indirect** **[XEQ] numérique**
* ° [ ]	Utilisation de la fonction secondaire d'une touche	
[ADV]	Avance papier	

Affichage et pile opérationnelle

° [APPEND]	Prolongation d'une chaîne alpha-numérique
* [ON]	Mise sous tension de l'affichage
[ENG] n	Notation ingénieur.n=nombre de chiffres significatifs après le premier
[FIX] n	Notation virgule fixe.n=nombre de chiffres après la virgule
[SCI] n	Notation scientifique.n=nombre de chiffres après la virgule
[EEX]	Introduction de l'exposant de 10
* ° [←]	Touche de correction
[ENTER↑]	Copie du contenu de X dans Y
[x⇆y] ou [X<>Y]	Echange des contenus de X et de Y
[X<>] n	Echange des contenus de X et d'un registre de stockage.n=numéro du registre

Encombrement en nombre d'octets	*Données*	*Sorties*	*Temps d'exécution*	*Message d'erreur*
1				
1				
2	x	conservé		NONEXISTENT
1				
2				
2	x,n	conservé	60	NONEXISTENT
3				
1				
1				
1				
2	Affichage		16	
2	Affichage		16	
2	Affichage		16	
1			84	
		*		
1	x	y,x	11	
1	x,y	y,x	10	
2	x			

R↓ ou RDN	Permutation circulaire des contenus de la pile vers le bas
R↑	Permutation circulaire des contenus de la pile vers le haut
CLD	Effacement de l'affichage
CLST	Effacement de la pile opérationnelle
CLx ou CLX	Effacement du registre X

Mathématique générale

+	Addition
−	Soustraction
÷ ou /	Division
x ou *	Multiplication
CHS	Changement de signe
ABS	Valeur absolue
INT	Partie entière
FRC	Partie fractionnaire
RND	Arrondi
$1/x$ ou 1/X	Inverse
x^2 ou X↑2	Carré
$\sqrt{x}$ ou SQRT	Racine carrée
%	Pourcentage
%CH	Différence en pourcentage
π ou PI	Utilisation du nombre π
MOD	Modulo
SIGN	Signe d'une expression

Logarithmes et exponentielles

LOG	Logarithme décimal
LN	Logarithme népérien
LN1+X	Logarithme népérien pour des arguments proches de 1
10^x ou 10↑X	Exponentielle décimale
e^x ou E↑X	Exponentielle népérienne
E↑ X-1	Exponentielle népérienne pour des arguments proches de 0
y^x ou Y↑X	Y élevé à la puissance x

Statistiques

ΣREG	Définition du bloc des six registres statistiques
Σ+	Accumulation de données
Σ-	Suppression ou correction de données
CLΣ	Effacement des registres statistiques
FACT	Factorielle
MEAN	Moyenne
SDEV	Ecart type

PROGRAMMER HP-41

Encombrement en nombre d'octets	*Données*	*Sorties*	*Temps d'exécution*	*Message d'erreur*
1	Pile OP	Pile OP	16	
1	Pile OP	Pile OP	12	
1			20	
1	Pile OP	T=Z=Y=X=Ø	10	
1	X	X=Ø	10	
1	X,Y	X,L	30	
1	X,Y	X,L	30	
1	X,Y	X,L	37	DATA ERROR
1	X,Y	X,L	37	
1	X	X	12	ou
1	X	X,L	13	ALPHA DATA
1	X	X,L	20	
1	X	X,L	20	ou
1	X	X,L	20	
1	X	X,L	37	OUT OF RANGE
1	X	X,L	41	
1	X	X,L	54	
1	X,Y	X,L	35	
1	X,Y	X,L	70	
1			19	
1	X,Y	X,L	20	
1	X	X,L	13	
1	X	X,L	260	
1	X	X,L	220	DATA ERROR
1	X	X,L	260	
				ALPHA DATA
1	X	X,L	100	
1	X	X,L	160	OUT OF RANGE
1	X	X,L	230	
1	X,Y	X,L	105	
2	affichage		30	NONEXISTENT
1	y,x	x,L	240	
1	y,x	x,L	250	OUT OF RANGE
1		REG	25	ALPHA DATA
1	x	x,L	20+4,2n	DATA ERROR
1		x	130	
1		x,y	480	

Trigonométrie

SIN	Sinus
COS	Cosinus
TAN	Tangente
SIN^{-1} ou ASIN	Arc sinus
COS^{-1} ou ACOS	Arc cosinus
TAN^{-1} ou ATAN	Arc tangente
DEG	Unité=degré
GRAD	Unité=grade
RAD	Unité=radian

Conversions

OCT	Décimal→octal
DEC	Octal→décimal
D-R	Degrés→radians
R-D	Radians→degrés
P-R	Polaire→rectangulaire
R-P	Rectangulaire→Polaire
HR	Sexagésimal→décimal
HMS	Décimal→sexagésimal
HMS+	Addition sexagésimale
HMS-	Soustraction sexagésimale

Registres mémoire

* SIZE n	Répartition de la mémoire.n=nombre de registres de stockage de données	
LAST x ou LASTX	Rappel du dernier nombre introduit	
STO n	Stockage.n=numéro du registre	**STO 00 à 15** **STO 16 à 99** **STO indirect**
STO + ou ST+ n	Addition en mémoire	
STO - ou ST- n	Soustraction en mémoire	n=numéro du registre
STO x ou STx n	Multiplication en mémoire	
STO ÷ ou ST÷ n	Division en mémoire	
RCL n	Rappel n=numéro du registre	**RCL 00 à 15** **RCL 16 à 99** **RCL indirect**
VIEW n	Contrôle d'un registre.n= numéro du registre	
CLRG	Effacement des registres	

Traitement des chaines ALPHA

AOFF	Annulation du mode ALPHA
AON	Mode alphanumérique
ASHF	Décalage à gauche de la chaîne ALPHA

Encombrement en nombre d'octets	*Données*	*Sorties*	*Temps d'exécution*	*Message d'erreur*
1	x	x,L	350	
1	x	x,L	450	ALPHA DATA
1	x	x,L	150	ou
1	x	x,L	450	DATA ERROR
1	x	x,L	450	OUT OF RANGE
1	x	x,L	310	
1			19	
1			20	
1			20	
1	x	x,L	120	
1	x	x,L	65	
1	x	x,L	110	ALPHA DATA
1	x	x,L	90	
1	x,y	x,y,L	300	ou
1	x,y	x,y,L	260	
1	x	x,L	40	DATA ERROR
1	x	x,L	30	
1	x	x,L	100	
1	x	x,L	100	
1	L	x	13	
2	x	REG	20	
2	x	REG		
2	x	REG		
2	x	REG	40	NONEXISTENT
1	REG	x	22	OUT OF RANGE
2	REG	x		
2	x	x		
2	affichage	affichage	130	
1		REG	12+2,7xSIZE	
1			31	
1			31	
1				

PROMPT	Affichage d'un message	
ASTO n	Stockage d'une chaîne ALPHA. n=numéro du registre	
ARCL n	Rappel d'une chaîne ALPHA. n=numéro du regisıre	
AVIEW	Contrôle d'une chaîne ALPHA	
CLA	Effacement du registre ALPHA	

Programmation

* ASN nom,n	Affectation.Nom de fonction et numéro de touche	
* BST	Un pas en arrière	
* CATALOG ou CAT n	Listage du catalogue.n=numéro du catalogue.	
* CLP nom	Effacement du programme nommé	
* COPY nom	Copie d'un programme	
* DEL n	Suppression de ligne de programme n=nombre de lignes.	
END	Fin de programme	
GTO n ou nom ou adresse indirecte	Branchement à l'adresse indiquée.	
* ° GTO . n ou nom	Branchement manuel	**GTO (00 à 14)** **GTO (15 à 99)** **GTO (ALPHA ajoute un octet par caractère du nom)** **GTO indirect**
* ° GTO . .	Branchement en fin de mémoire programme et préparation pour un nouveau programme	
LBL n ou nom	Adresse d'un programme dans la mémoire	**LBL (00 à 14)** **LBL (15 à 99)** **LBL (ALPHA ajoute un octet par caractère du nom)**
* PACK	Compactage de la mémoire programme	
PSE	Pause	
RTN	Renvoi	
R/S	Marche/arrêt du programme	
* SST	Un pas en avant	
STOP	Arrêt programmé	

PROGRAMMER HP-41

Encombrement en nombre d'octets	*Données*	*Sorties*	*Temps d'exécution*	*Message d'erreur*
1		REG		NONEXISTENT
2		REG		
2				
1			280	
1			9	
				PRIVATE-RAM
				NONEXISTENT
2			32	ROM
3				
2				
2				
2				
1				
2				
4				
1			1300	
1				
1				

Indicateurs binaires

CF n ou adresse indirecte	Annule l'indicateur concerné
FC? n ou adresse indirecte	Test pour 0 de l'indicateur concerné
FS?C n ou adresse indirecte	Test pour 0 de l'indicateur concerné et mise à zéro
FS? n ou adresse indirecte	Test pour 1 de l'indicateur concerné
FC?C n ou adresse indirecte	Test pour 1 de l'indicateur concerné et mise à zéro
SF n ou adresse indirecte	Mise à 1 de l'indicateur concerné

Tests

x=y?	ou	X=Y?	X est-il égal à Y ?
x=0?	ou	X=0?	X est-il égal à O ?
x>y?	ou	X>Y?	X est-il supérieur à Y ?
x>0?	ou	X>0?	X est-il supérieur à O ?
x<y?	ou	X<Y?	X est-il inférieur à Y ?
x<0?	ou	X<0?	X est-il inférieur à O ?
x≤y?	ou	X<=Y?	X est-il inférieur ou égal à Y ?
x≤0?	ou	X<=0?	X est-il inférieur ou égal à O ?
x≠y?	ou	X≠Y?	X est-il différent de Y ?
x≠0?	ou	X≠0?	X est-il différent de O ?

DSE n	Soustraction d'une unité et saut si égalité. n = numéro du registre.
ISG n	Addition d'une unité et saut si plus grand. n = numéro du registre.

Encombrement en nombre d'octets	*Données*	*Sorties*	*Temps d'exécution*	*Message d'erreur*
2		drapeau n	31	
2			22-35	
2		drapeau n	22-35	
2			22-35	
2		drapeau n	33	
2		drapeau n	31	
1	x,y		10-21	
1	x		12-23	
1	x,y		28-37	
1	x		12-23	
1	x,y			
1	x			
1	x,y		28-38	
1	x		12-23	
1	x,y		10-21	
1	x		12-23	
2	n	n		
2	n	n		NONEXISTENT

ANNEXE 4

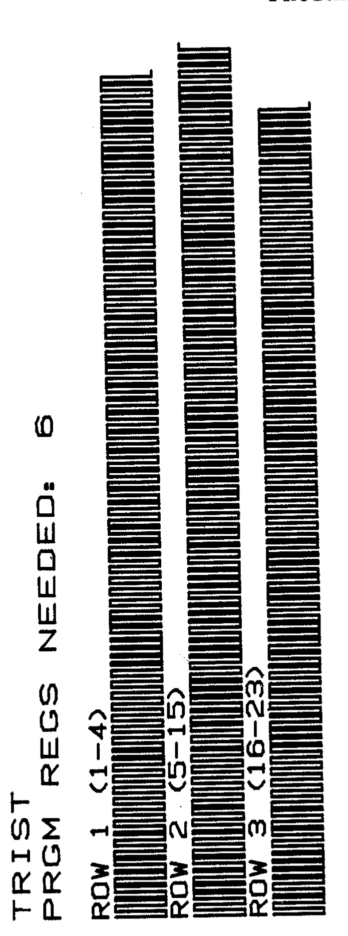
Drapeaux

N°	signification	fonctions d'accès	État à la mise sous tension	Modifié par hp-41
0-10	Défini par l'utilisateur	sf cf fs ?c fc ?c	Conservé par la mémoire permanente	non
11	Exécution automatique des programmes		Désarmé	
12-17	Utilisé par les périphériques			Variable
18-20	Non encore défini			Non
21	Arrêt d'exécution avec view et aview		Variable	Oui prend l'état du flag 55
22	Une valeur numérique a été entrée au clavier		Désarmé	oui
23	Idem 22 pour chaîne de caractères			
24	Inhibe arrêt en cas de dépassement de capacité $+/-9.999999999E99$ pris par défaut. Sauf division par zéro			Non, n'est pas désarmé par R/S contrairemen à ce que dit le manuel p164

N°	signification	fonctions d'accès	État à la mise sous tension	Modifié par hp-41
25	Inhibe l'arrêt en cas d'erreur (instruction non exécutée	sf cf fs ?c fc ?c	Désarmé	Désarmé à la première erreur
26	Signal sonore actif		Armé	Non
27	Mode user		Mémoire permanente	
28	Point décimal ou virgule			
29	Séparateur de groupe de trois chiffres			
30	Catalogue en cours	Aucune	Désarmé	Désarmé pour l'utilisateur
31-35	Périphérique HP-IL		?	Non
36-41	Format d'affichage	Fix sci eng	Mémoire permanente	
42-43	Mode trigonométrique	Deg rad grad		
44	Inhibe l'auto-extinction	On	Désarmé	
45	Entrée de donnée	Aucune		Toujours désarmé pour l'utilisateur
46	Entrée de fonction			
47	Shift			
48	Alpha	Aon aoff		Désarmé par touche prgm
49	Batterie		Variable	Oui
50	Message à l'affichage	View aview cld	Désarmé	Toujours désarmé pour l'utilisateur
51	Sst en cours	Aucune		
52	Mode programme	Prgm		
53	Fonction I/O	Aucune		
54	Pause en cours			
55	Imprimante thermique présente		Variable	Oui

ANNEXE 5

Procédures et code-barre



TRIST

SIZE= 000

BUT

CLASSE PAR ORDRE
CROISSANT OU DECROISSANT
4 NOMBRES CONTENUS
DANS LA PILE.

OBSERVATION

LE FLAG 00 PERMET
D'INVERSER L'ORDRE
DU TRI.
(F00=CLEAR → X<=Y<=Z<=T)

PILE

ETAT INITIAL	ETAT FINAL
T=	T=
Z=	Z=
Y=	Y=
X=	X=
L= L	L= L

REGISTRES

AUCUN

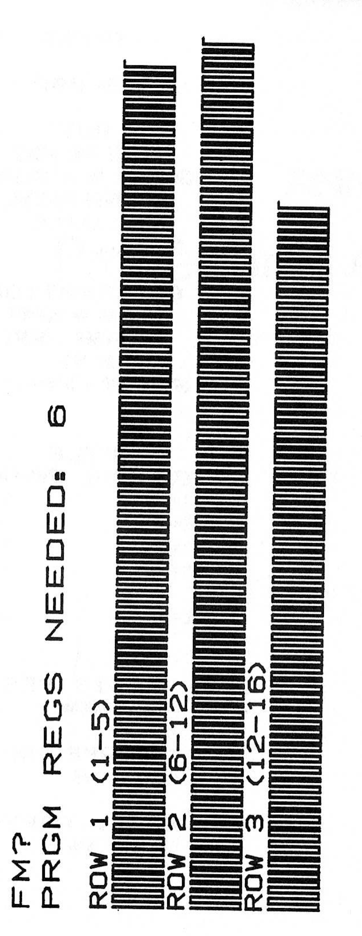
DRAPEAUX

00

PROC. APPEL

NON

```
01♦LBL "TRIST"
02 X>Y?
03 X<>Y
04 X<> T
05 X>Y?
06 X<>Y
07 X<> Z
08 X>Y?
09 X<>Y
10 R↑
11 X>Y?
12 X<>Y
13 X<> T
14 X>Y?
15 X<>Y
16 R↑
17 X>Y?
18 X<>Y
19 FC? 00
20 RTN
21 X<> Z
22 R↑
23 .END.
```



FM?

SIZE= 000

BUT

FOURNIR EN X LE NOMBRE DE DECIMALES SELECTIONNEES.

OBSERVATION

PILE

ETAT INITIAL	ETAT FINAL
T= d	T= c
Z= c	Z= b
Y= b	Y= a
X= a	X= Nb de dec.
L= L	L= 8

REGISTRES

AUCUN

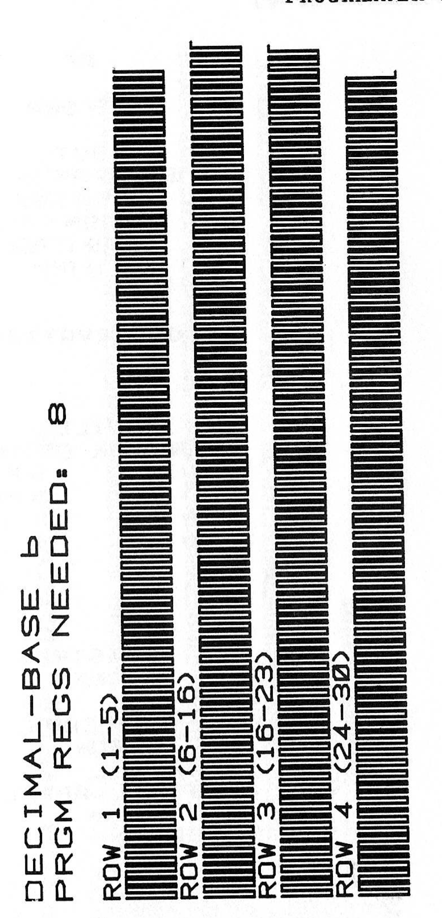
DRAPEAUX

AUCUN

PROC. APPEL

NON

```
01♦LBL "FM?"
02 .
03 SIGN
04 FS? 39
05 ST+ L
06 ST+ X
07 FS? 38
08 ST+ L
09 ST+ X
10 FS? 37
11 ST+ L
12 ST+ X
13 FS? 36
14 ST+ L
15 X<> L
16 .END.
```



10-b

SIZE= 000

BUT

CONVERTIR UN NOMBRE DE LA BASE 10 EN BASE b.

OBSERVATION

b<=99

PILE

ETAT INITIAL	ETAT FINAL
T= T	T= 10↑I
Z= Z	Z= 0
Y= BASE	Y= BASE
X= Nb 10	X= Nb b
L= L	L= 10

REGISTRES

AUCUN

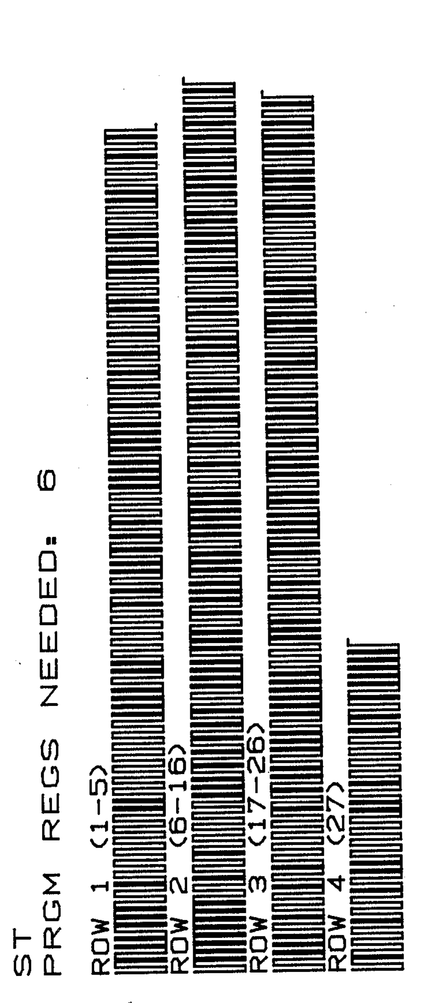
DRAPEAUX

AUCUN

PROC. APPEL

NON

```
01♦LBL "10-b"
02 1
03 .
04 RDN
05 RDN
06♦LBL 01
07 X<>Y
08 /
09 LASTX
10 X<>Y
11 INT
12 ST- L
13 X<>Y
14 ST* L
15 R↑
16 ST* L
17 X<> L
18 ST+ T
19 CLX
20 10
21 X<Y?
22 ST* X
23 ST* L
24 X<> L
25 RDN
26 X<>Y
27 X≠0?
28 GTO 01
29 X<> Z
30 .END.
```



ST

SIZE= 000

BUT

CALCUL DES COEFFICIENTS
ENTIERS S ET T TELS QUE
SY + TX = PGCD (a,b)

OBSERVATION

PILE

ETAT INITIAL	ETAT FINAL
T= T	T= a''
Z= Z	Z= b'
Y= a	Y= S
X= b	X= T
L= L	L= 0

REGISTRES

AUCUN

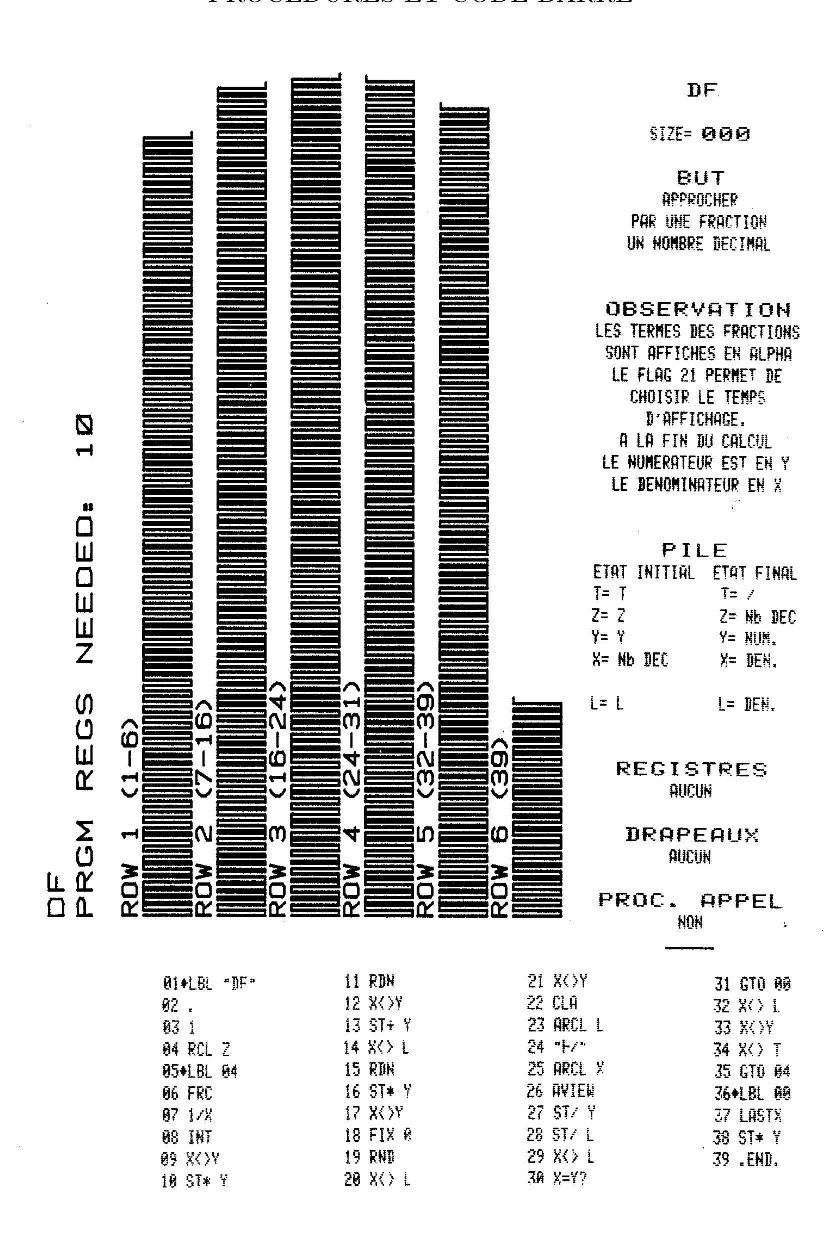
DRAPEAUX

AUCUN

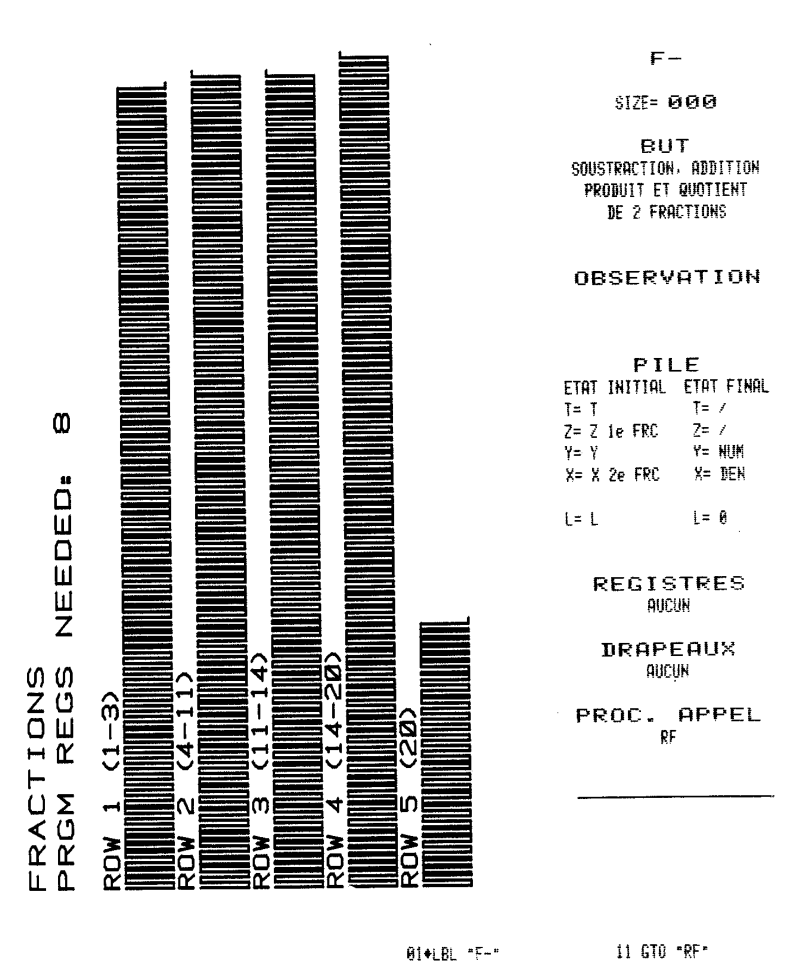
PROC. APPEL

RF

```
01♦LBL "ST"
02 XEQ "RF"
03
04 ENTER↑
05♦LBL 03
06 RDN
07 CHS
08 X<=0?
09 DSE X
10 ""
11 STO T
12 X<>Y
13 *
14 LASTX
15 RDN
16 CHS
17 ISG X
18 ""
19 X<>Y
20 /
21 LASTX
22 RDN
23 FRC
24 X≠0?
25 GTO 03
26 X<> L
27 .END.
```




```
01♦LBL "DF"
02 .
03 1
04 RCL Z
05♦LBL 04
06 FRC
07 1/X
08 INT
09 X<>Y
10 ST* Y
11 RDN
12 X<>Y
13 ST+ Y
14 X<> L
15 RDN
16 ST* Y
17 X<>Y
18 FIX 0
19 RND
20 X<> L
21 X<>Y
22 CLA
23 ARCL L
24 "⊢/"
25 ARCL X
26 AVIEW
27 ST/ Y
28 ST/ L
29 X<> L
30 X=Y?
31 GTO 00
32 X<> L
33 X<>Y
34 X<> T
35 GTO 04
36♦LBL 00
37 LASTX
38 ST* Y
39 .END.
```



F-

SIZE= 000

BUT

SOUSTRACTION, ADDITION
PRODUIT ET QUOTIENT
DE 2 FRACTIONS

OBSERVATION

PILE

ETAT INITIAL	ETAT FINAL
T= T	T= /
Z= Z 1e FRC	Z= /
Y= Y	Y= NUM
X= X 2e FRC	X= DEN
L= L	L= 0

REGISTRES
AUCUN

DRAPEAUX
AUCUN

PROC. APPEL
RF

```
01♦LBL "F-"
02 CHS
03♦LBL "F+"
04 ST* T
05 X<> Z
06 ST* Z
07 *
08 RCL Z
09 +
10 X<>Y
11 GTO "RF"
12♦LBL "F/"
13 X<>Y
14♦LBL "F*"
15 ST* Z
16 RDN
17 ST* Z
18 RDN
19 GTO "RF"
20 .END.
```



DIV

SIZE= 000

BUT

CALCUL LE QUOTIENT EXACT
ET LE RESTE

OBSERVATION

CONSERVE LES REG. Z ET T

PILE

ETAT INITIAL	ETAT FINAL
T= T	T= T
Z= Z	Z= Z
Y= DIVIDENDE	Y= RESTE
X= DIVISEUR	X= Q. EX.
L= L	L= .5

REGISTRES

AUCUN

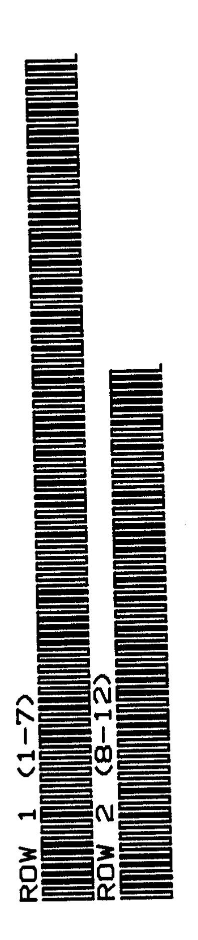
DRAPEAUX

AUCUN

PROC. APPEL

NON

```
01♦LBL "DIV"
02 ST/ Y
03 X<>Y
04 INT
05 ST- L
06 X<>Y
07 ST* L
08 CLX
09 .5
10 ST+ L
11 X<> L
12 INT
13 X<>Y
14 .END.
```



CB

SIZE= 000

BUT

CALCUL DU NOMBRE
DE COMBINAISONS DE
K OBJETS PARMI N

OBSERVATION

PILE

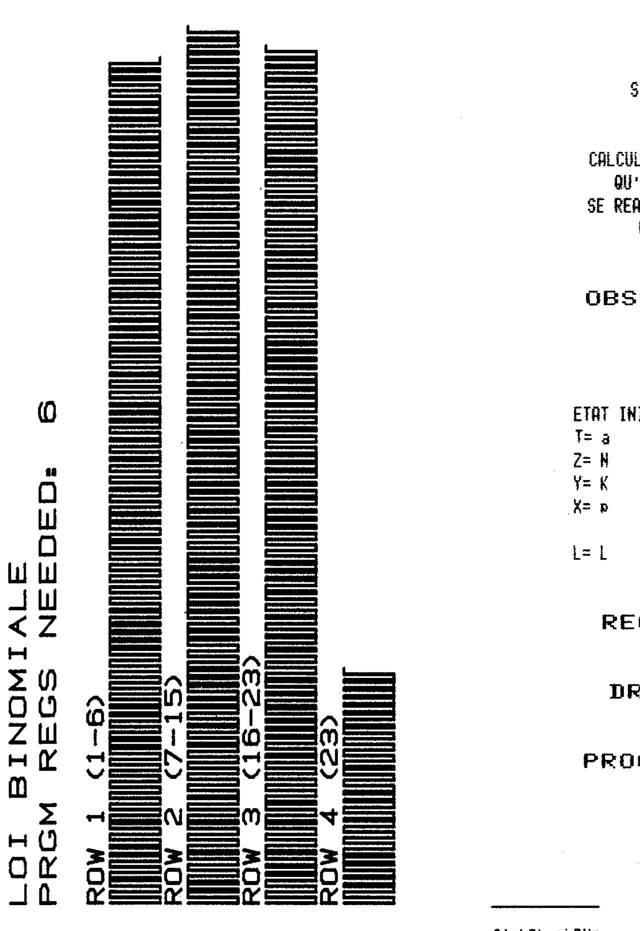
ETAT INITIAL	ETAT FINAL
T= b	T= a
Z= a	Z= a
Y= N	Y= a
X= K	X= C
L= L	L= /

REGISTRES
AUCUN

DRAPEAUX
AUCUN

PROC. APPEL
NON

```
01♦LBL "CB"
02 X<>Y
03 FACT
04 X<>Y
05 ST- L
06 LASTX
07 FACT
08 X<>Y
09 FACT
10 *
11 /
12 .END.
```



LBN

SIZE= 000

BUT

CALCUL LA PROBABILITE
QU'UN EVENEMENT
SE REALISE K FOIS SUR
N EPREUVES

OBSERVATION

PILE

ETAT INITIAL	ETAT FINAL
T= a	T= P
Z= N	Z= P
Y= K	Y= P
X= p	X= P
L= L	L= 1

REGISTRES

AUCUN

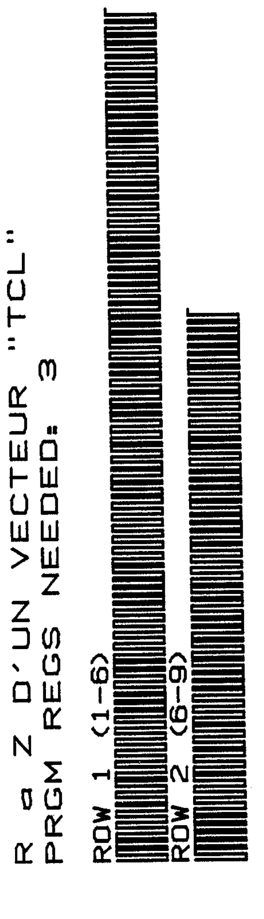
DRAPEAUX

AUCUN

PROC. APPEL

CB

```
01◆LBL "LBN"
02 1
03 X<>Y
04 -
05 LASTX
06 X<> Z
07 ST- T
08 X<> T
09 Y↑X
10 LASTX
11 X<> Z
12 R↑
13 ST+ T
14 Y↑X
15 LASTX
16 RDN
17 *
18 STO T
19 RDN
20 X<>Y
21 XEQ "CB"
22 *
23 .END.
```



TCL

SIZE= VARIABLE

BUT

EFFACER LES MEMOIRES
DESIGNEES PAR LE CODE DE
CONTROLE DU VECTEUR
PLACE EN X

OBSERVATION

PILE

ETAT INITIAL	ETAT FINAL
T= T	T= T
Z= Z	Z= Z
Y= Y	Y= Y
X= ddd,fffii	X= 0
L= L	L= /

REGISTRES

CEUX DESIGNES EN X

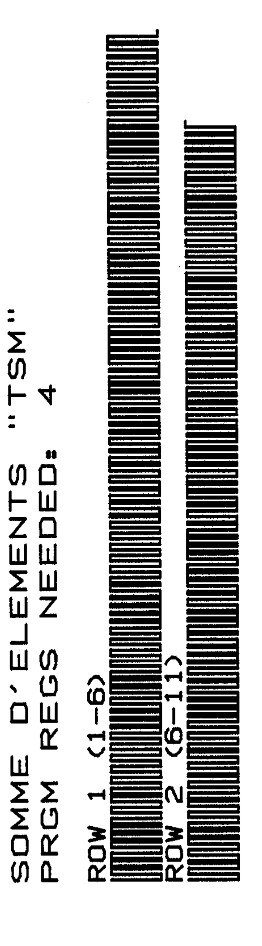
DRAPEAUX

AUCUN

PROC. APPEL

NON

```
01♦LBL "TCL"
02 SIGN
03 CLX
04♦LBL 01
05 STO IND L
06 ISG L
07 GTO 01
08 RDN
09 .END.
```



TSM

SIZE= VARIABLE

BUT

CALCUL LA SOMME
ALGEBRIQUE DES ELEMENTS
D'UN VECTEUR DONT
LE CODE DE CONTROLE EST
INITIALEMENT EN X

OBSERVATION

PILE

ETAT INITIAL	ETAT FINAL
T= T	T= T
Z= Z	Z= Z
Y= Y	Y= Y
X= CODE	X= SOMME
L= L	L= /

REGISTRES
AUCUN

DRAPEAUX
AUCUN

PROC. APPEL
NON

```
01♦LBL "TSM"
02 SIGN
03 CLX
04♦LBL 02
05 RCL IND L
06 ST+ Y
07 RDN
08 ISG L
09 GTO 02
10 ISG L
11 .END.
```

ADRESSE DU MAXIMUM "TMX"
PRGM REGS NEEDED: 6

ROW 1 (1-6)
ROW 2 (6-16)
ROW 3 (17-22)

TMX

SIZE= VARIABLE

BUT

RECHERCHE DANS UN VECTEUR DONT LE CODE DE CONTROLE EST EN X, LA VALEUR MAXIMALE ET L'ADRESSE DE CETTE VALEUR.

OBSERVATION

PILE

ETAT INITIAL	ETAT FINAL
T= T	T= /
Z= Z	Z= MAX
Y= Y	Y= Y
X= CODE	X= ADRESSE
L= L	L= 0

REGISTRES

AUCUN

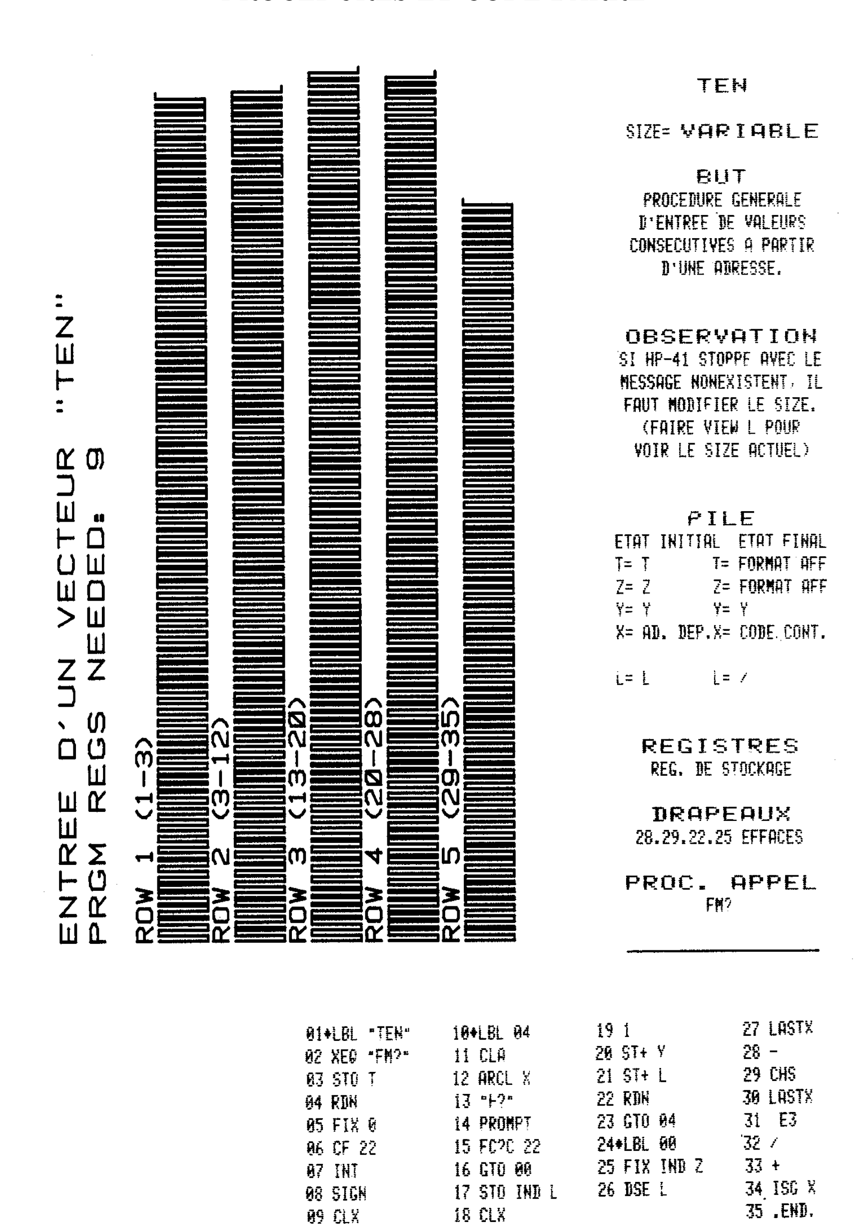
DRAPEAUX

AUCUN

PROC. APPEL

NON

```
01♦LBL "TMX"
02 X<>Y
03 SIGN
04 RDN
05 RCL X
06 RCL IND X
07 ENTER↑
08♦LBL 03
09 RDN
10 RCL IND Y
11 X<=Y?
12 GTO 00
13 X<>Y
14 LASTX
15 +
16♦LBL 00
17 ISG Z
18 GTO 03
19 CLX
20 LASTX
21 R↑
22 .END.
```



ENTREE D'UN VECTEUR "TEN"
PRGM REGS NEEDED: 9

ROW 1 (1-3)
ROW 2 (3-12)
ROW 3 (13-20)
ROW 4 (20-28)
ROW 5 (29-35)

TEN

SIZE= VARIABLE

BUT
PROCEDURE GENERALE D'ENTREE DE VALEURS CONSECUTIVES A PARTIR D'UNE ADRESSE.

OBSERVATION
SI HP-41 STOPPE AVEC LE MESSAGE NONEXISTENT, IL FAUT MODIFIER LE SIZE. (FAIRE VIEW L POUR VOIR LE SIZE ACTUEL)

PILE

ETAT INITIAL	ETAT FINAL
T= T	T= FORMAT AFF
Z= Z	Z= FORMAT AFF
Y= Y	Y= Y
X= AD. DEP.	X= CODE CONT.
L= L	L= /

REGISTRES
REG. DE STOCKAGE

DRAPEAUX
28.29.22.25 EFFACES

PROC. APPEL
FM?

```
01♦LBL "TEN"      10♦LBL 04         19 1              27 LASTX
02 XEQ "FM?"      11 CLA            20 ST+ Y          28 -
03 STO T          12 ARCL X         21 ST+ L          29 CHS
04 RDN            13 "⊢?"           22 RDN            30 LASTX
05 FIX 0          14 PROMPT         23 GTO 04         31  E3
06 CF 22          15 FC?C 22        24♦LBL 00         32 /
07 INT            16 GTO 00         25 FIX IND Z      33 +
08 SIGN           17 STO IND L      26 DSE L          34 ISG X
09 CLX            18 CLX                              35 .END.
```

TRI DES VALEURS "TRI"
PRGM REGS NEEDED: 5

ROW 1 (1-4)

ROW 2 (4-10)

ROW 3 (11)

TRI

SIZE= VARIABLE

BUT

RANGE PAR ORDRE DECROISSANT LES ELEMENTS D'UN VECTEUR DONT LE CODE EST EN X.

OBSERVATION

PILE

ETAT INITIAL	ETAT FINAL
T= T	T= /
Z= Z	Z= /
Y= Y	Y= /
X= ddd,fffii	X= .
L= L	L= /

REGISTRES

CEUX DESIGNES

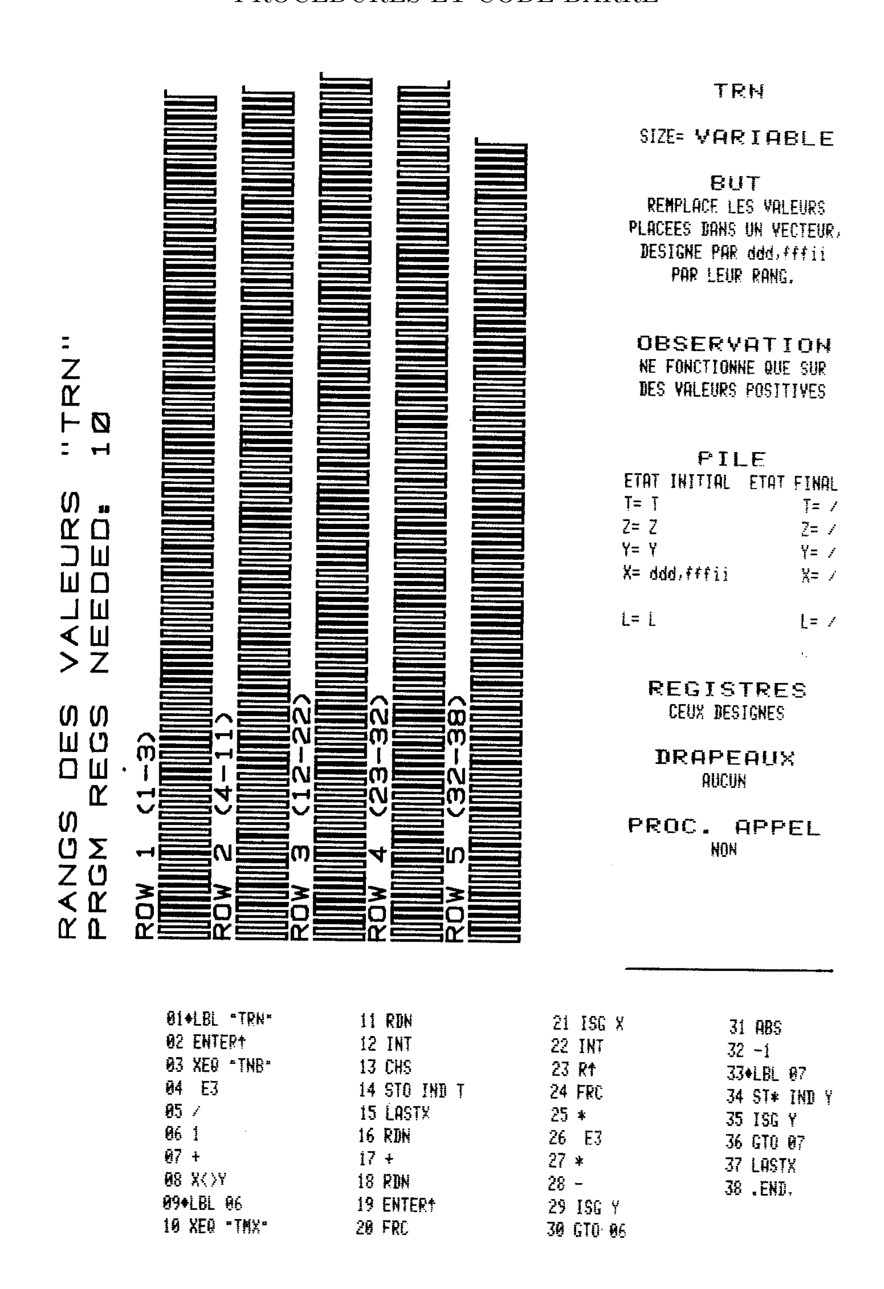
DRAPEAUX

AUCUN

PROC. APPEL

TMX

```
01♦LBL "TRI"
02♦LBL 05
03 ENTER↑
04 XEQ "TMX"
05 RCL IND X
06 X<> IND Z
07 STO IND Y
08 RCL Z
09 ISG X
10 GTO 05
11 .END.
```



RANGS DES VALEURS "TRN"
PRGM REGS NEEDED: 10

ROW 1 (1-3)
ROW 2 (4-11)
ROW 3 (12-22)
ROW 4 (23-32)
ROW 5 (32-38)

TRN

SIZE= VARIABLE

BUT

REMPLACE LES VALEURS PLACEES DANS UN VECTEUR, DESIGNE PAR ddd,fffii PAR LEUR RANG.

OBSERVATION

NE FONCTIONNE QUE SUR DES VALEURS POSITIVES

PILE

ETAT INITIAL	ETAT FINAL
T= T	T= /
Z= Z	Z= /
Y= Y	Y= /
X= ddd,fffii	X= /
L= L	L= /

REGISTRES

CEUX DESIGNES

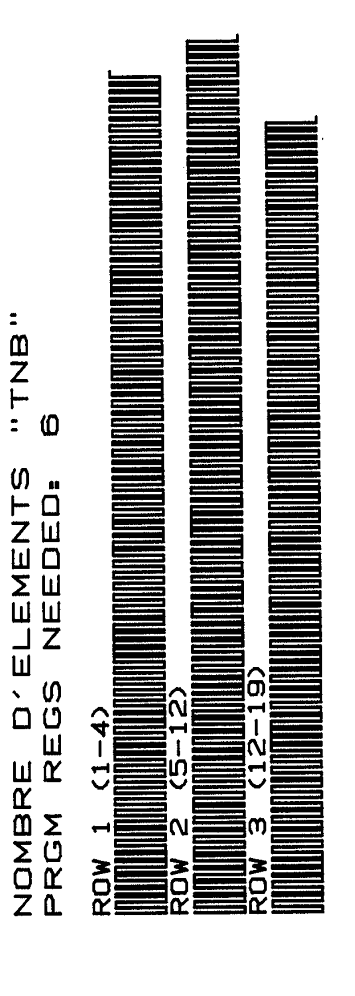
DRAPEAUX

AUCUN

PROC. APPEL

NON

```
01♦LBL "TRN"      11 RDN            21 ISG X          31 ABS
02 ENTER↑         12 INT            22 INT            32 -1
03 XEQ "TNB"      13 CHS            23 R↑             33♦LBL 07
04  E3            14 STO IND T      24 FRC            34 ST* IND Y
05 /              15 LASTX          25 *              35 ISG Y
06 1              16 RDN            26  E3            36 GTO 07
07 +              17 +              27 *              37 LASTX
08 X<>Y           18 RDN            28 -              38 .END.
09♦LBL 06         19 ENTER↑         29 ISG Y
10 XEQ "TMX"      20 FRC            30 GTO 06
```



TNB

SIZE= 000

BUT

CALCUL DU NOMBRE
D'ELEMENTS D'UN VECTEUR
DONT LE CODE ddd,fffii
EST EN X.

OBSERVATION

NE VERIFIE PAS
L'EXISTENCE DU VECTEUR.

PILE

ETAT INITIAL	ETAT FINAL
T= T	T= Z
Z= Z	Z= Z
Y= Y	Y= Y
X= ddd,fffii	X= N
L= L	L= /

REGISTRES

AUCUN

DRAPEAUX

AUCUN

PROC. APPEL

NON

```
01♦LBL "TNB"
02 INT
03 ST- L
04  E-3
05 ST* Y
06 RDN
07 ST- L
08 X<> L
09 ISG X
10 INT
11 ST- L
12 ST/ L
13 X<> L
14  E3
15 *
16 INT
17 1
18 +
19 .END.
```

MELANGE ALEATOIRE "TML"
PRGM REGS NEEDED: 5

ROW 1 (1-4)

ROW 2 (4-10)

ROW 3 (11)

TML

SIZE= VARIABLE

BUT

EFFECTUE UN BRASSAGE
PSEUDO-ALEATOIRE
DES VALEURS DU VECTEUR

OBSERVATION

PILE

ETAT INITIAL	ETAT FINAL
T= T	T= /
Z= Z	Z= /
Y= Y	Y= /
X= ddd.fffii	X= /
L= L	L= /

REGISTRES
"EVENTUELLEMENT ?!"

DRAPEAUX
AUCUN

PROC. APPEL
GAG

```
01♦LBL "TML"
02♦LBL 08
03 ENTER↑
04 XEQ "GAG"
05 RCL IND X
06 X<> IND Z
07 STO IND Y
08 RCL Z
09 ISG X
10 GTO 08
11 .END.
```



TAF

SIZE= VARIABLE

BUT

AFFICHE LES ELEMENTS
DU VECTEUR

OBSERVATION

UTILISE LE FLAG 21 POUR
OBTENIR L'ARRET A
CHAQUE VALEUR.

PILE

ETAT INITIAL	ETAT FINAL
T= T	T= Z
Z= Z	Z= Z
Y= Y	Y= Y
X= ddd.fffii	X= X
L= L	L= N

REGISTRES

AUCUN

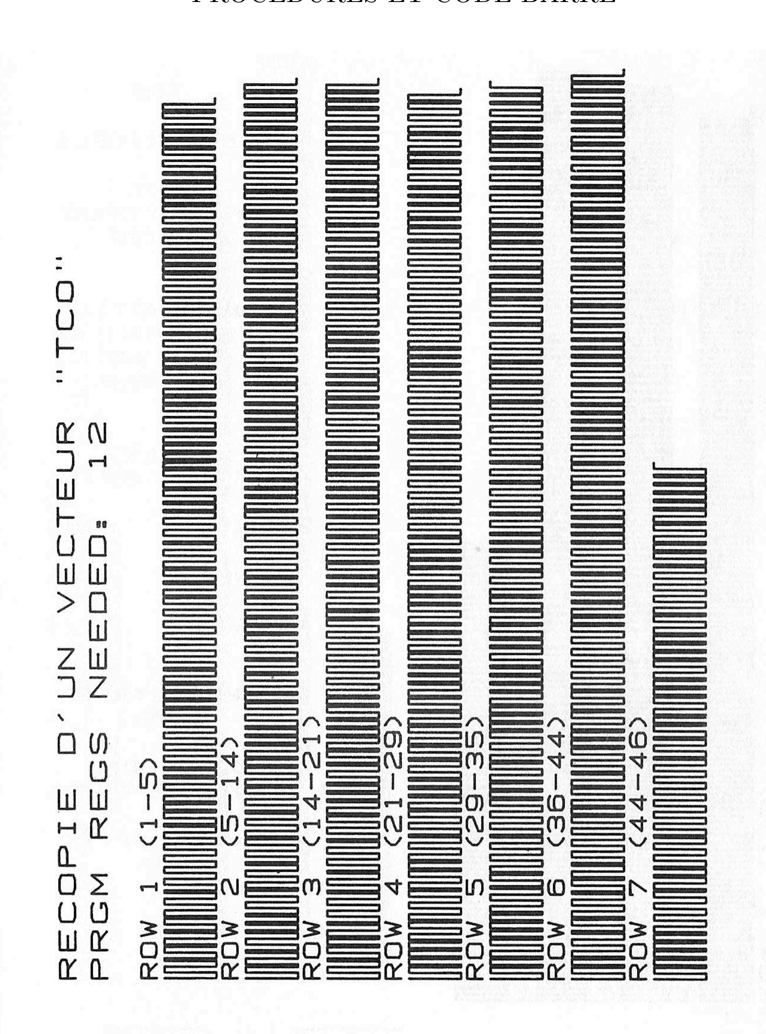
DRAPEAUX

36 A 39 TESTES RESTITUES

PROC. APPEL

FM?

```
01♦LBL "TAF"
02 XEQ "FM?"
03 RDN
04 SIGN
05 CLX
06♦LBL 09
07 CLA
08 FIX 0
09 ARCL X
10 "⊢= "
11 FIX IND T
12 ARCL IND L
13 AVIEW
14 ISG X
15 ""
16 ISG L
17 GTO 09
18 ST- L
19 X<> L
20 .END.
```


RECOPIE D'UN VECTEUR "TCO"
PRGM REGS NEEDED: 12
ROW 1 (1-5)
ROW 2 (5-14)
ROW 3 (14-21)
ROW 4 (21-29)
ROW 5 (29-35)
ROW 6 (36-44)
ROW 7 (44-46)

TCO

SIZE= VARIABLE

BUT

RECOPIE UN VECTEUR DONT
LE CODE EST EN X
A PARTIR DE L'ADRESSE
PLACEE EN Y

OBSERVATION

IL PEUT Y AVOIR
RECOUVREMENT DU VECTEUR
INITIAL PAR LE VECTEUR
FINAL.

PILE

ETAT INITIAL	ETAT FINAL
T= T	T= /
Z= Z	Z= /
Y= ddd'	Y= /
X= ddd,fffiii	X= ddd,FFF
L= L	L= /

REGISTRES

CEUX NECESSAIRES A
LA COPIE

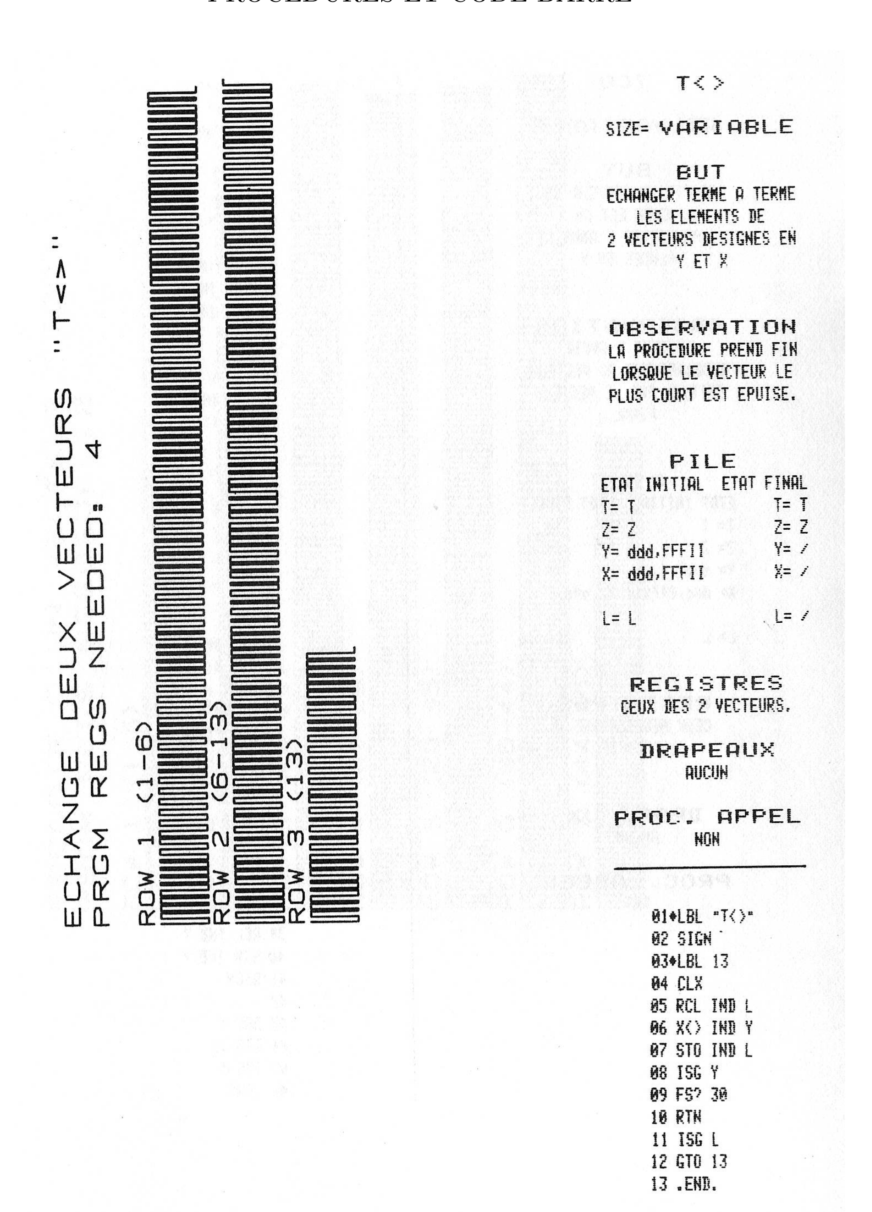
DRAPEAUX

AUCUN

PROC. APPEL

TNB

```
01♦LBL "TCO"
02 X<=Y?
03 GTO 00
04 X<>Y
05  E-3
06 X<>Y
07 ST* Y
08 +
09♦LBL 10
10 RCL IND Y
11 STO IND Y
12 SIGN
13 +
14 ISG Y
15 GTO 10
16♦LBL 11
17 INT
18 ST- L
19  E-3
20 ST* Y
21 ST- Y
22 ST/ L
23 X<> L
24 +
25 RTN
26♦LBL 00
27 XEQ 11
28 X<>Y
29 RCL Y
30 XEQ "TNB"
31 +
32 2
33 -
34  E-3
35 X<>Y
36 ST* Y
37 +
38♦LBL 12
39 RCL IND Y
40 STO IND Y
41 SIGN
42 -
43 DSE Y
44 GTO 12
45 ISG X
46 .END.
```



T<>

SIZE= VARIABLE

BUT

ECHANGER TERME A TERME
LES ELEMENTS DE
2 VECTEURS DESIGNES EN
Y ET X

OBSERVATION

LA PROCEDURE PREND FIN
LORSQUE LE VECTEUR LE
PLUS COURT EST EPUISE.

PILE

ETAT INITIAL	ETAT FINAL
T= T	T= T
Z= Z	Z= Z
Y= ddd,FFFII	Y= /
X= ddd,FFFII	X= /
L= L	L= /

REGISTRES

CEUX DES 2 VECTEURS.

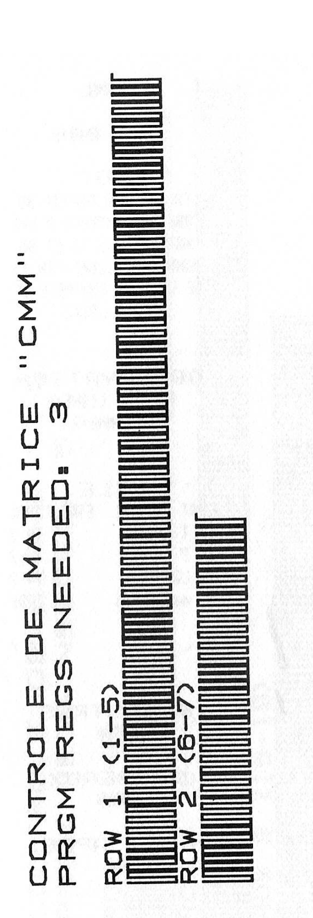
DRAPEAUX

AUCUN

PROC. APPEL

NON

```
01♦LBL "T<>"
02 SIGN
03♦LBL 13
04 CLX
05 RCL IND L
06 X<> IND Y
07 STO IND L
08 ISG Y
09 FS? 30
10 RTN
11 ISG L
12 GTO 13
13 .END.
```



CMM

SIZE= 000

BUT

CALCULER LE CODE DE CONTROLE PERMETTANT DE BALAYER L'ENSEMBLE DES ELEMENTS D'UNE MATRICE.

OBSERVATION

PILE

ETAT INITIAL	ETAT FINAL
T= T	T= Z
Z= Z	Z= Z
Y= Y	Y= Y
X= dddfffii	X= ddd,fff
L= L	L= /

REGISTRES
AUCUN

DRAPEAUX
AUCUN

PROC. APPEL
NON

```
01◆LBL "CMM"
02  E3
03 *
04 INT
05  E3
06 /
07 .END.
```

CONTROLE DE COLONNE "CMC"
PRGM REGS NEEDED: 2

ROW 1 (1-5)

CML

SIZE= 000

BUT

CALCULER A PARTIR DU CODE DE CONTROLE D'UNE MATRICE (EN X) ET DU NUMERO DE LIGNE (EN Y) LE CODE DE CONTROLE DE CETTE LIGNE.

OBSERVATION

ORIGINE LIGNE=0
COLONNE=0

PILE

ETAT INITIAL	ETAT FINAL
T= T	T= Z
Z= Z	Z= Z
Y= LGN	Y= Z
X= ddd,fffii	X= CODE
L= L	L= /

REGISTRES

AUCUN

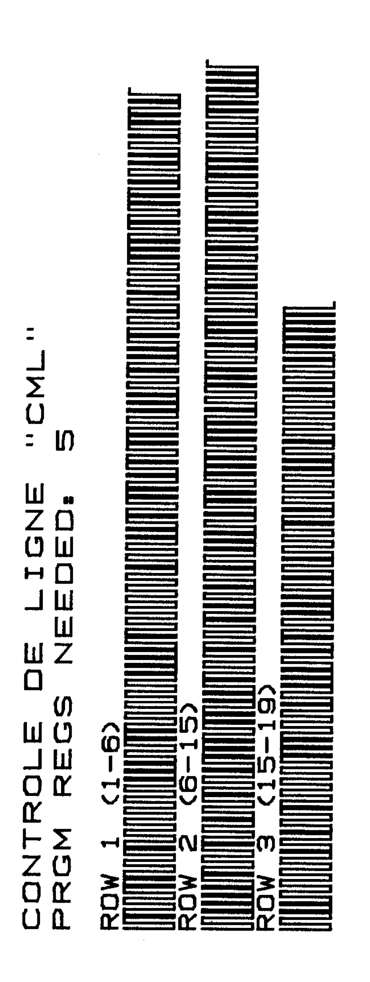
DRAPEAUX

AUCUN

PROC. APPEL

NON

```
01+LBL "CMC"
02 X<>Y
03 INT
04 +
05 .END.
```



CMC

SIZE= 000

BUT

CALCULER A PARTIR DU CODE DE CONTROLE (EN X) ET DU NUMERO (EN Y) LE CODE DE CONTROLE DE LA COLONNE.

OBSERVATION

PILE

ETAT INITIAL	ETAT FINAL
T= T	T= T
Z= Z	Z= T
Y= COL	Y= Z
X= ddd,fffii	X= CODE COL
L= L	L= COL

REGISTRES

AUCUN

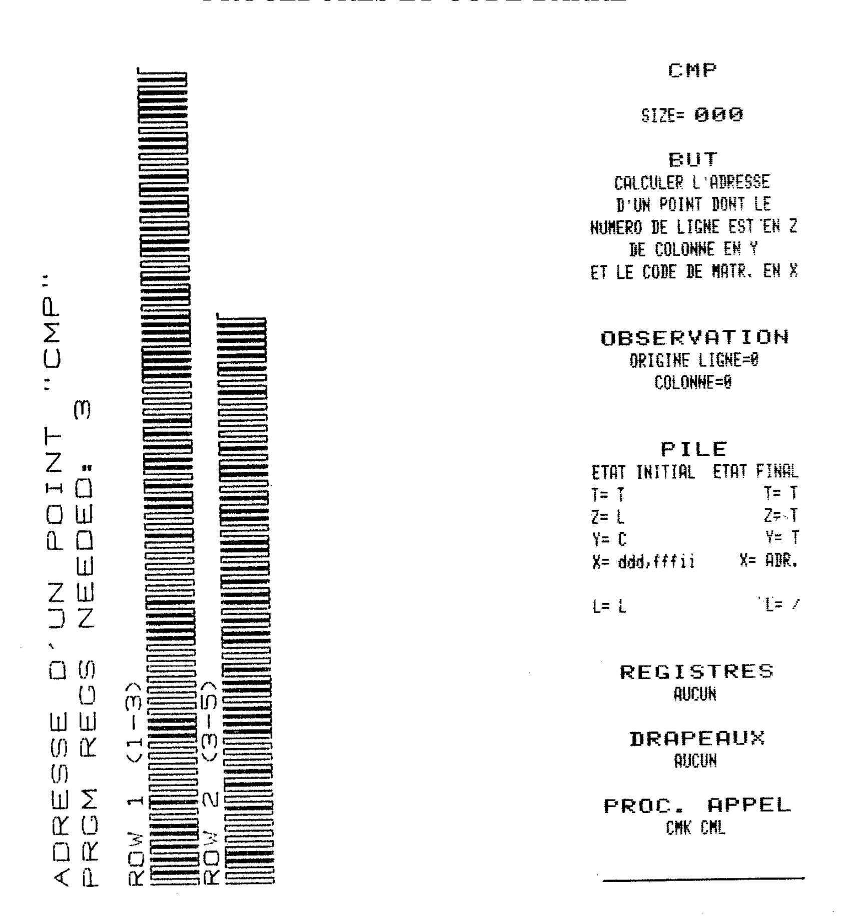
DRAPEAUX

AUCUN

PROC. APPEL

NON

```
01♦LBL "CML"
02 X<>Y
03 INT
04 RCL Y
05 FRC
06 ISG X
07 INT
08 *
09 ST+ Y
10 X<> L
11 X<>Y
12 INT
13 ST+ Y
14 X<>Y
15  E-3
16 ST* Y
17 -
18 +
19 .END.
```



CMP

SIZE= 000

BUT
CALCULER L'ADRESSE
D'UN POINT DONT LE
NUMERO DE LIGNE EST EN Z
DE COLONNE EN Y
ET LE CODE DE MATR. EN X

OBSERVATION
ORIGINE LIGNE=0
COLONNE=0

PILE

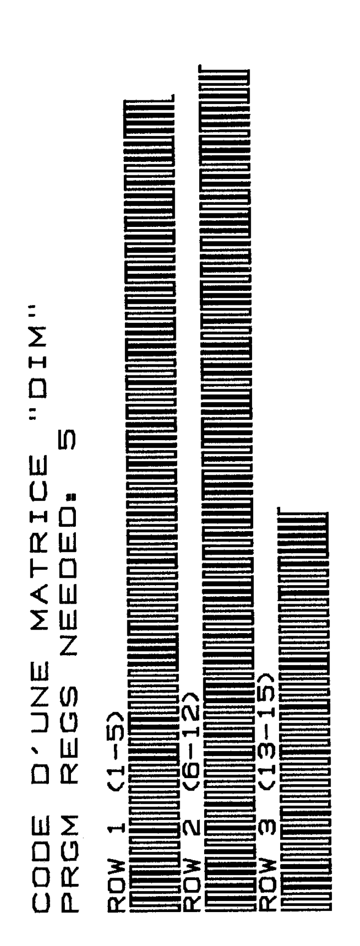
ETAT INITIAL	ETAT FINAL
T= T	T= T
Z= L	Z= T
Y= C	Y= T
X= ddd,fffii	X= ADR.
L= L	L= /

REGISTRES
AUCUN

DRAPEAUX
AUCUN

PROC. APPEL
CMK CML

```
01•LBL "CMP"
02 XEQ "CMC"
03 XEQ "CML"
04 INT
05 .END.
```



DIM

SIZE= 000

BUT

OBTENIR LE CODE DE CONTROLE D'UNE MATRICE DONT LA BASE EST EN Z, LE NOMBRE DE LIGNES EN Y ET LE NOMBRE DE COLONES EN X.

OBSERVATION PILE

ETAT INITIAL	ETAT FINAL
T= T	T= T
Z= BASE	Z= T
Y= Nb LGN	Y= T
X= Nb CLN	X= ddd,FFFII
L= /	L= /

REGISTRES

AUCUN

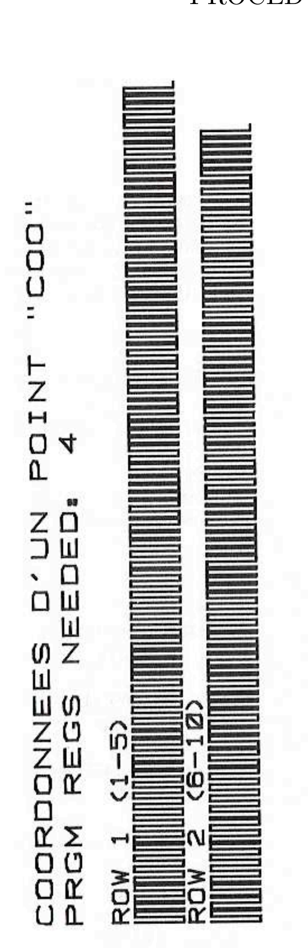
DRAPEAUX

AUCUN

PROC. APPEL

NON

```
01♦LBL "DIM"
02 *
03  E5
04 ST/ L
05 CLX
06 RCL Z
07 ST+ Y
08 DSE Y
09 CLX
10  E3
11 ST/ Y
12 X<> L
13 +
14 +
15 .END.
```



COO

SIZE= 000

BUT

CALCULER LES COORDONNEES LIGNE, COLONNE, D'UNE VALEUR DONT L'ADRESSE EST PLACEE EN Y, LE CODE MATRICE EN X.

OBSERVATION

PILE

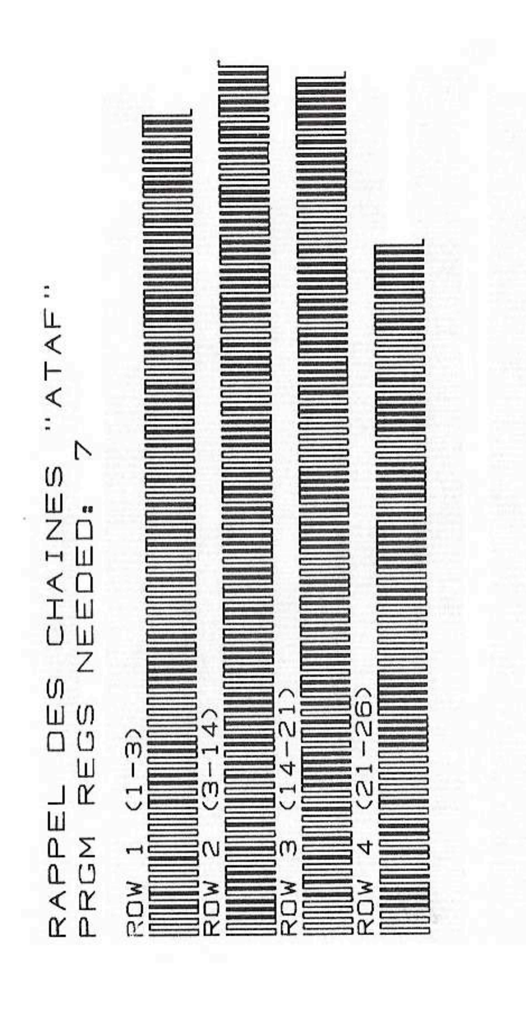
ETAT INITIAL	ETAT FINAL
T= T	T= T
Z= Z	Z= Z
Y= ADR.	Y= L
X= CODE MATR.	X= C
L= L	L= /

REGISTRES
AUCUN

DRAPEAUX
AUCUN

PROC. APPEL
DIV

```
01♦LBL "COO"
02 INT
03 ST- Y
04 X<> L
05 FRC
06 ISG X
07 INT
08 XEQ "DIV"
09 X<>Y
10 .END.
```



ATAF

SIZE= VARIABLE

BUT

AFFICHAGE DES CHAINES ALPHA EN MEMOIRE.

OBSERVATION

PILE

ETAT INITIAL	ETAT FINAL
T= T	T= /
Z= Z	Z= /
Y= Y	Y= /
X= CODE MATR.	X= /
L= L	L= /

REGISTRES

AUCUN

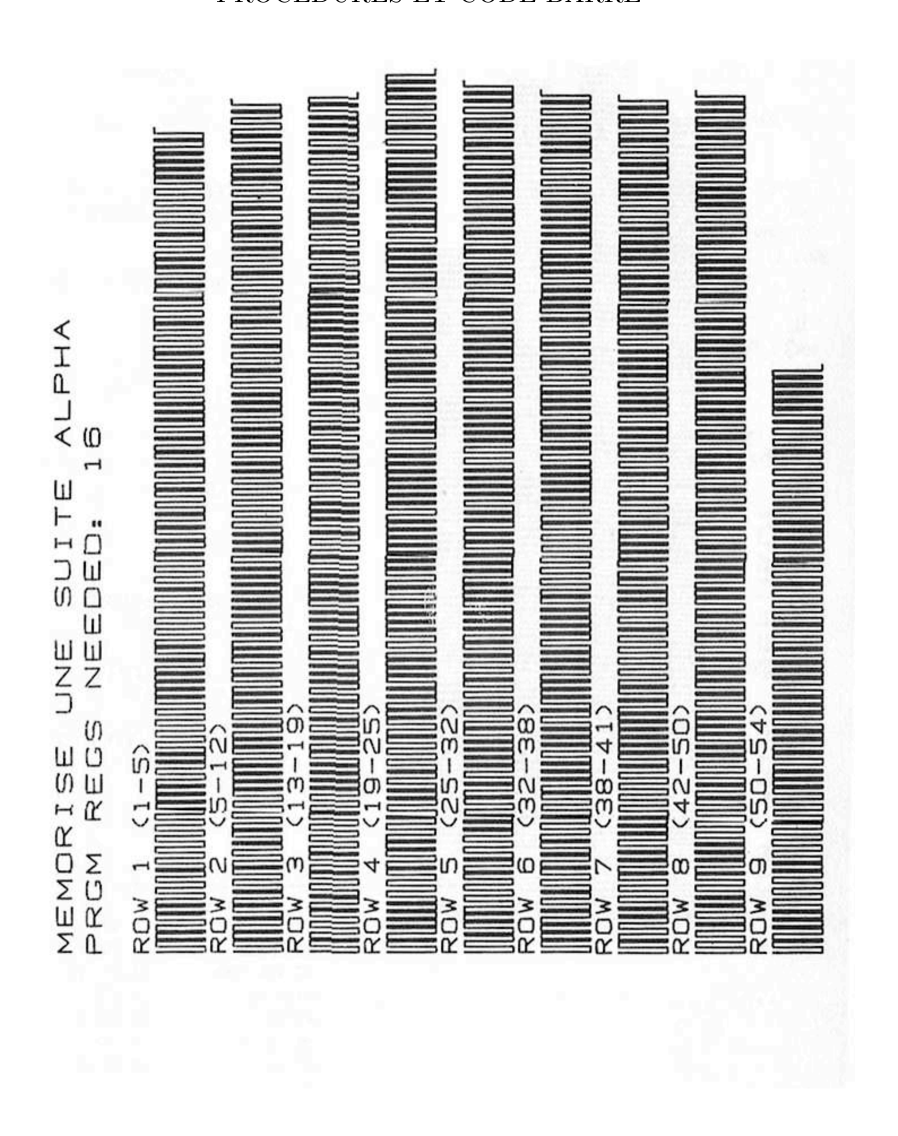
DRAPEAUX

AUCUN

PROC. APPEL

CMM

```
01♦LBL "ATAF"
02 ENTER↑
03 XEQ "CMM"
04 X<>Y
05 FRC
06 ISG X
07 INT
08 SIGN
09♦LBL 13
10 CLA
11 CLX
12 LASTX
13♦LBL 12
14 ARCL IND Y
15 ISG Y
16 FS? 30
17 GTO 11
18 DSE X
19 GTO 12
20 AVIEW
21 TONE 7
22 GTO 13
23♦LBL 11
24 AVIEW
25 TONE 7
26 .END.
```


MEMORISE UNE SUITE ALPHA
PRGM REGS NEEDED: 16
ROW 1 (1-5)
ROW 2 (5-12)
ROW 3 (13-19)
ROW 4 (19-25)
ROW 5 (25-32)
ROW 6 (32-38)
ROW 7 (38-41)
ROW 8 (42-50)
ROW 9 (50-54)

ATEN

SIZE= VARIABLE

BUT

PLACER EN MEMOIRE UNE SUCCESSION DE CHAINES

OBSERVATION

PILE

ETAT INITIAL	ETAT FINAL
T= T	T= /
Z= Z	Z= /
Y= Nb MAX de CARACT.	Y= /
X= ADR. DEP.	X= CODE MAT.
L= L	L= /

REGISTRES

REG. DE STOCKAGE

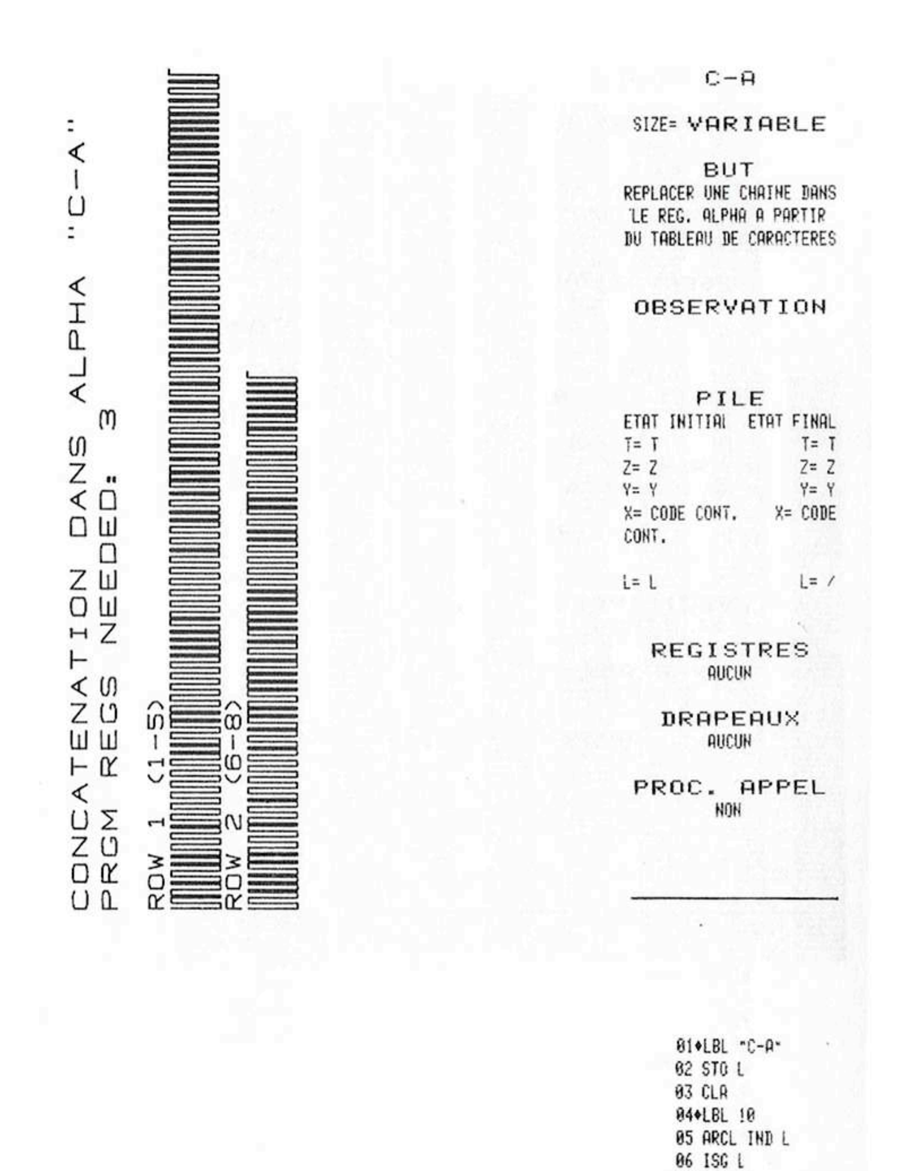
DRAPEAUX

23 EFFACE

PROC. APPEL

FM?

```
01♦LBL "ATEN"
02 .1
03 %
04 +
05 XEQ "FM?"
06 FIX 0
07 X<> Z
08 1
09 -
10 6
11 /
12 INT
13 SIGN
14 ST+ L
15 CLX
16 STO T
17 CF 23
18♦LBL 14
19 "CHN "
20 ARCL T
21 "⊢?"
22 AON
23 PROMPT
24 FC?C 23
25 GTO "CODE"
26 CLX
27 LASTX
28♦LBL 15
29 ASTO IND Y
30 ASHF
31 ISG Y
32 STO X
33 DSE X
34 GTO 15
35 ISG T
36 STO X
37 GTO 14
38♦LBL "CODE"
39 FIX IND Z
40 X<> L
41  E2
42 /
43 X<>Y
44 INT
45 ST- L
46 ST+ Y
47 CLX
48  E3
49 ST* L
50 ST/ Y
51 X<> L
52 +
53 AOFF
54 .END.
```



CONCATENATION DANS ALPHA "C-A"
PRGM REGS NEEDED: 3

ROW 1 (1-5)

ROW 2 (6-8)

C-A

SIZE= VARIABLE

BUT

REPLACER UNE CHAINE DANS LE REG. ALPHA A PARTIR DU TABLEAU DE CARACTERES

OBSERVATION

PILE

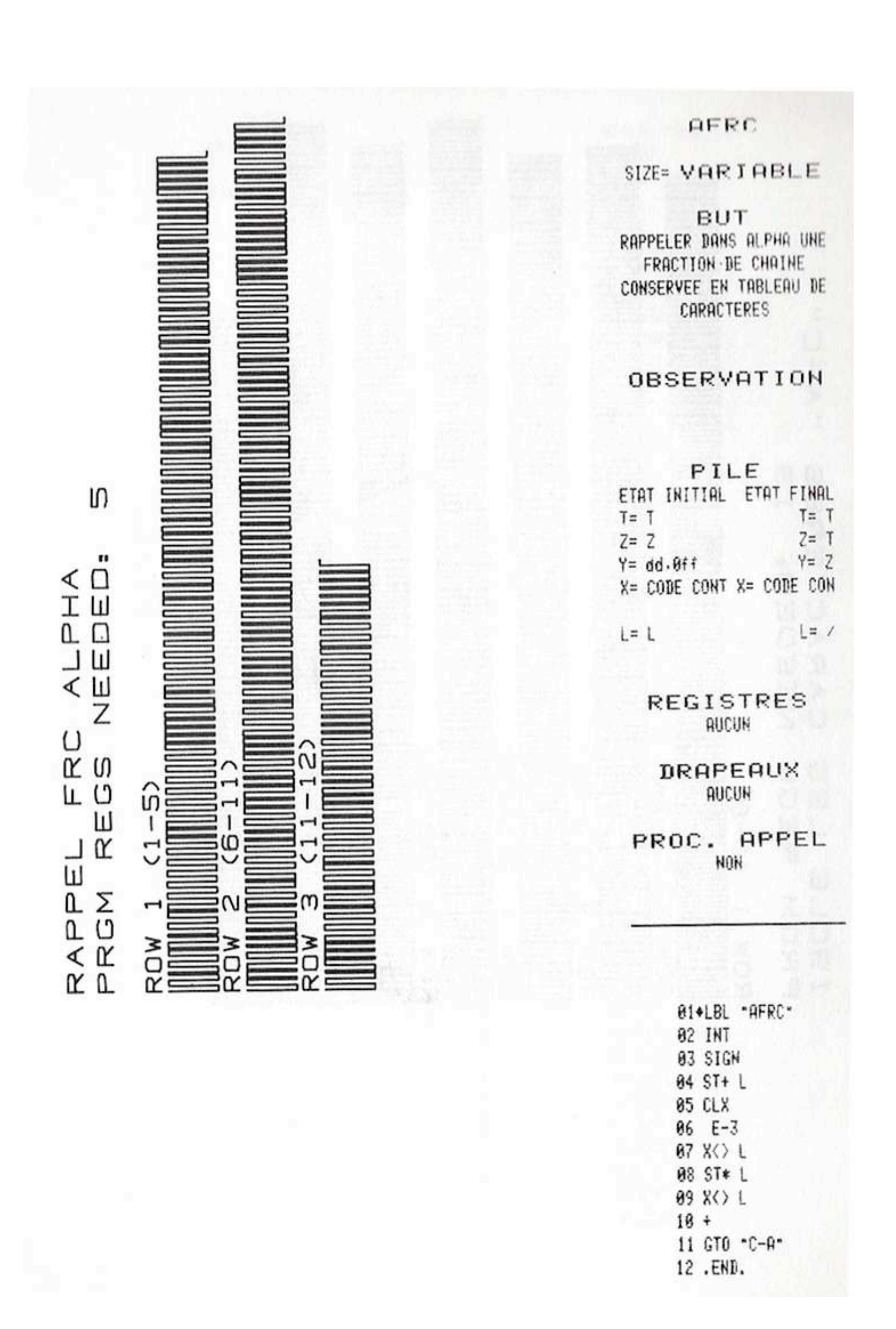
ETAT INITIAL	ETAT FINAL
T= T	T= T
Z= Z	Z= Z
Y= Y	Y= Y
X= CODE CONT.	X= CODE CONT.
L= L	L= /

REGISTRES
AUCUN

DRAPEAUX
AUCUN

PROC. APPEL
NON

```
01♦LBL "C-A"
02 STO L
03 CLA
04♦LBL 10
05 ARCL IND L
06 ISG L
07 GTO 10
08 .END.
```



AFRC

SIZE= VARIABLE

BUT

RAPPELER DANS ALPHA UNE FRACTION DE CHAINE CONSERVEE EN TABLEAU DE CARACTERES

OBSERVATION

PILE

ETAT INITIAL	ETAT FINAL
T= T	T= T
Z= Z	Z= T
Y= dd.0ff	Y= Z
X= CODE CONT	X= CODE CON
L= L	L= /

REGISTRES

AUCUN

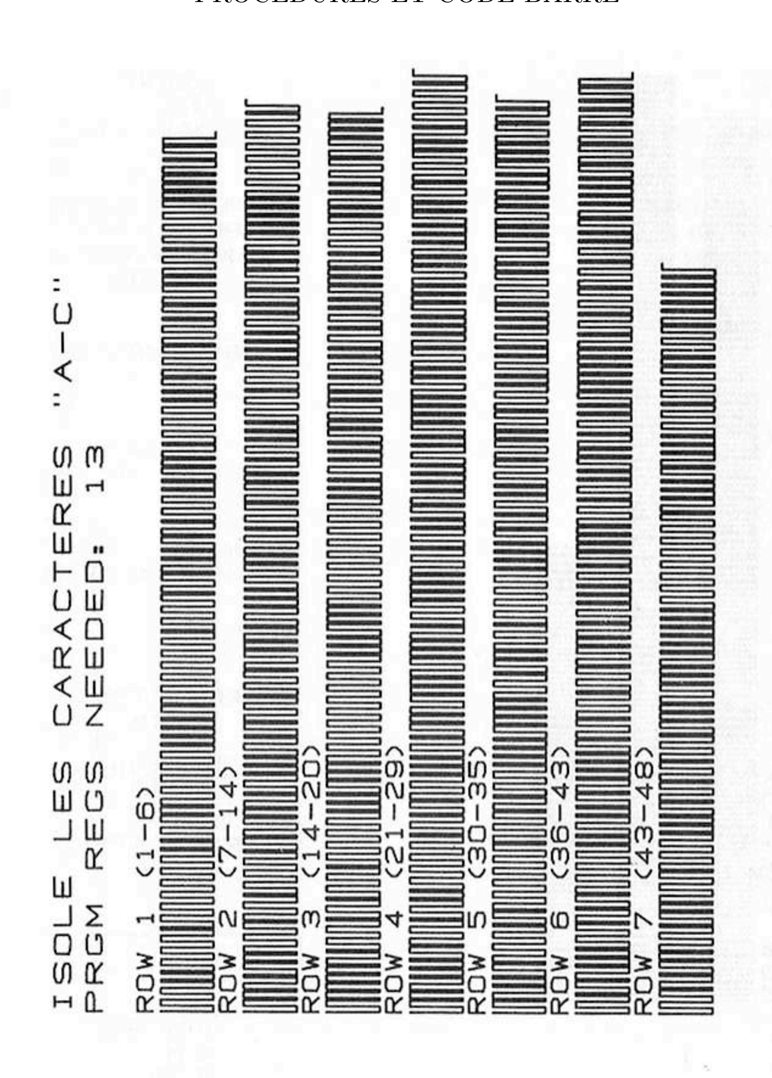
DRAPEAUX

AUCUN

PROC. APPEL

NON

```
01♦LBL "AFRC"
02 INT
03 SIGN
04 ST+ L
05 CLX
06  E-3
07 X<> L
08 ST* L
09 X<> L
10 +
11 GTO "C-A"
12 .END.
```


ISOLE LES CARACTERES "A-C"
PRGM REGS NEEDED: 13
ROW 1 (1-6)
ROW 2 (7-14)
ROW 3 (14-20)
ROW 4 (21-29)
ROW 5 (30-35)
ROW 6 (36-43)
ROW 7 (43-48)

A-C

SIZE= VARIABLE

BUT

DECOMPOSER LE REGISTRE
ALPHA POUR CONSTITUER
UN VECTEUR DE CARACTERES

OBSERVATION

SOYEZ PATIENT
18 CARACTERES MAX

PILE

ETAT INITIAL	ETAT FINAL
T= T	T= /
Z= Z	Z= /
Y= Y	Y= /
X= ADR. DEP.	X= CODE CONT
L= L	L= /

REGISTRES

REG. STOCKAGE

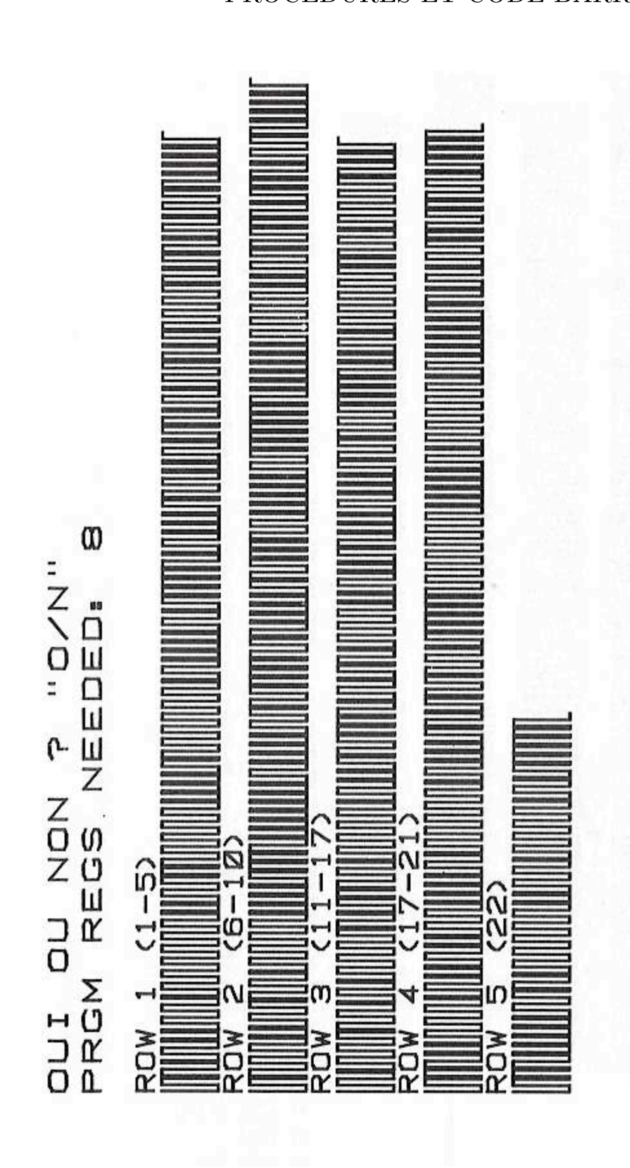
DRAPEAUX

AUCUN

PROC. APPEL

NON

```
01♦LBL "A-C"
02 INT
03 .1
04 %
05 +
06 SIGN
07 ASTO Y
08 ASHF
09 ASTO Z
10 ASHF
11 ASTO T
12 CLA
13 ASTO X
14 XEQ 14
15 XEQ 14
16 XEQ 14
17 LASTX
18 FRC
19 ST- L
20  E3
21 ST* Y
22 X<> L
23 X=Y?
24 ASTO IND X
25 X≠Y?
26 DSE X
27 LASTX
28 /
29 +
30 RTN
31♦LBL 14
32 X=Y?
33 GTO 00
34 "12345"
35 ARCL Y
36 ASTO IND L
37 ASHF
38 ASTO Y
39 "6"
40 ARCL IND L
41 ASHF
42 ASTO IND L
43 ISG L
44 STO X
45 GTO 14
46♦LBL 00
47 RDN
48 .END.
```


OUI OU NON ? "O/N"
PRGM REGS NEEDED: 8
ROW 1 (1-5)
ROW 2 (6-10)
ROW 3 (11-17)
ROW 4 (17-21)
ROW 5 (22)

O/N

SIZE= 000

BUT

INTERROGATION SUR UNE ALTERNATIVE

OBSERVATION

UTILISABLE SANS PRECAUTION PARTICULIERE

PILE

ETAT INITIAL	ETAT FINAL
T= T	T= T
Z= Z	Z= Z
Y= Y	Y= Y
X= X	X= X
L= L	L= /

REGISTRES

AUCUN

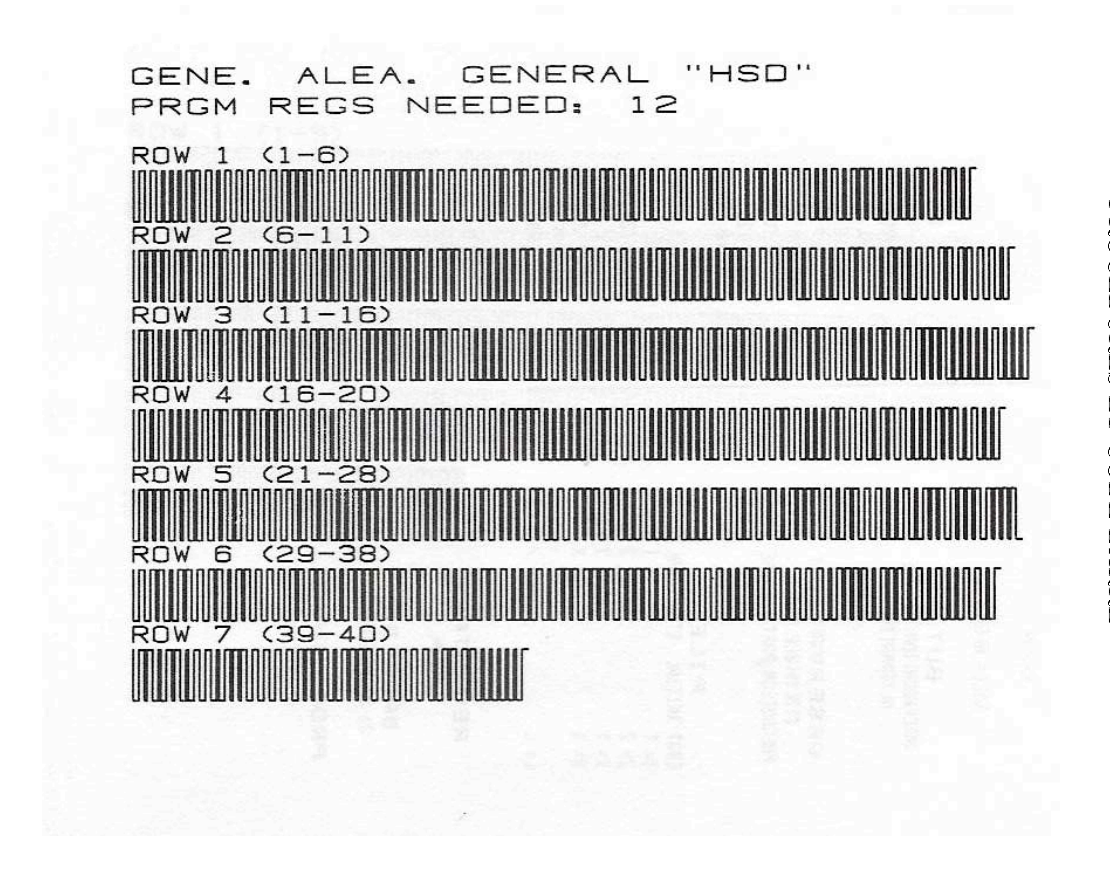
DRAPEAUX

21 23 25 EFFACES

PROC. APPEL

NON

```
01♦LBL "O/N"
02 AON
03 CF 21
04♦LBL 01
05 CF 23
06 "⊢(O/N)?"
07 AVIEW
08♦LBL 00
09 PSE
10 FC? 23
11 GTO 00
12 ASTO L
13 SF 25
14 GTO IND L
15 CLA
16 GTO 01
17♦LBL "N"
18 CF 23
19♦LBL "O"
20 CF 25
21 AOFF
22 .END.
```


GENE. ALEA. GENERAL "HSD"
PRGM REGS NEEDED: 12
ROW 1 (1-6)
ROW 2 (6-11)
ROW 3 (11-16)
ROW 4 (16-20)
ROW 5 (21-28)
ROW 6 (29-38)
ROW 7 (39-40)

HSD-GAG

SIZE= 001

BUT

GENERATEUR DE NOMBRES
ALEATOIRES

OBSERVATION

HSD EST APPELE EN DEBUT
DE PROGRAMME.
IL "ENSEMENCE"
LE GENERATEUR.
GAG UTILISE LE REG. X
POUR DEFINIR LES BORNES

PILE

ETAT INITIAL	ETAT FINAL
T= T	T= /
Z= Z	Z= Z
Y= Y	Y= Y
X= nnn,mmm	X= N
L= L	L= /

REGISTRES

00

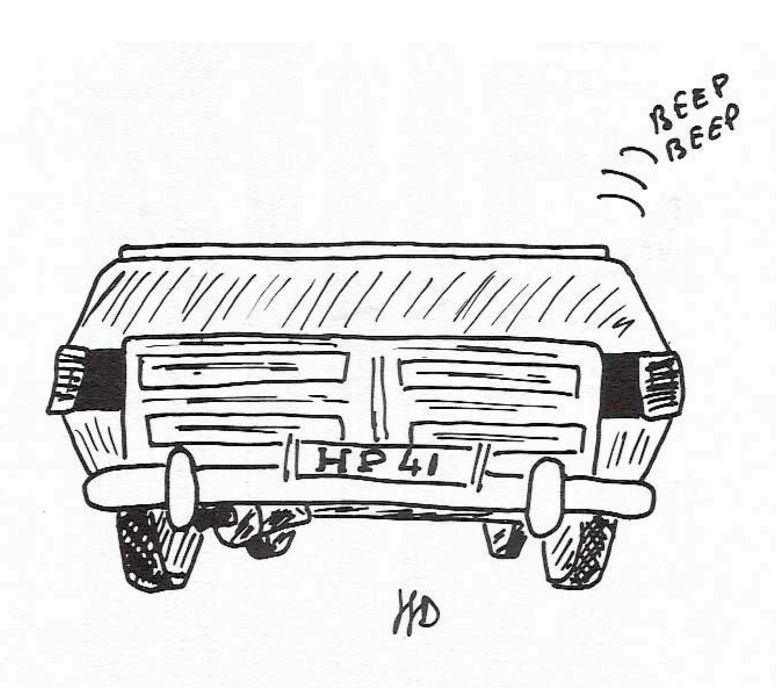
DRAPEAUX

AUCUN

PROC. APPEL

NON

```
01♦LBL "HSD"
02 DEG
03 RCL 00
04 SF 25
05 FRC
06 FS?C 25
07 X=0?
08 FS? 30
09 GTO 00
10 RDN
11 "ALEA 0<X<1?"
12 TONE 9
13 PROMPT
14 STO 00
15 RDN
16 GTO "HSD"
17♦LBL 00
18 CLX
19♦LBL "GAG"
20 FRC
21 ST- L
22 E3
23 ST* Y
24 X<> L
25 X>Y?
26 X<>Y
27 -
28 ISG X
29 STO X
30 RCL 00
31 ST* Y
32 X<> L
33 +
34 INT
35 RCL 00
36 ACOS
37 FRC
38 STO 00
39 RDN
40 .END.
```



Achevé d'imprimer en septembre 1982
sur les presses de l'imprimerie Laballery et Cie
58500 Clamecy
Dépôt légal : septembre 1982

n° d'impression : 20831
N° d'édition : 86595-56-1
ISBN : 2-86595-56-5



programmer HP. 41

Ce 1er volume étudie HP-41, sans ses périphériques, selon quatre axes : LES TESTS ET LES DRAPEAUX : obtenir les fonctions manquantes (X ≥ 0) ou logiques (et, ou, non...). LA PILE OPERATIONNELLE : utiliser pleinement la notation polonaise grâce au logigraphe. LES TABLEAUX NUMERIQUES : traiter une colonne aussi aisément qu'une ligne, trier, ranger... LES CHAINES DE CARACTERES : extraire un passage d'un texte, le découper ou le reconstituer. Une quarantaine de nouvelles fonctions, fournies sous forme de code barre, les index et les tableaux rassemblés en annexe constituent un outil de référence permanent.

EDITIONS DU P.S.I.
41-51, rue jacquard - BP 86
France - 77400 lagny s/marne
ISBN : 2.86595.056.5
imprimé en France